EXHIBIT 2

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

software products. A development effort of this scope and complexity would be an extremely large project, very aggressive, and of high-risk to be pursued within this timeframe. It would be exceedingly difficult for a project of this magnitude to be successfully completed within a 24 month period, but equally difficult for business reasons (e.g., pursuing a time sensitive market opportunity) for the development effort to exceed 24 months.

Based on the required level of business and technical knowledge and expected attrition, however, it would be tenuous to retain a team of this size and caliber for the required duration within a single U.S. city. While there are a limited number of U.S. cities that possess a large enough, technically qualified talent pool, these same cities house a number of established software development shops which actively compete for the best technical resources. As a whole, these circumstances highlight why my cost estimate is particularly conservative in light of the constraints at issue. Infringement, rather than independent development, would save not only the costs of development identified through my Function Point and COCOMO analyses, but the significant time and risk associated with independent development.

Further, testimony from this litigation reflects the additional value that would come from hiring personnel with experience through former employment by JD Edwards, PeopleSoft, Siebel, and/or Oracle.[2] The desire to hire an even smaller subset of available personnel, namely, personnel with experience in similar roles at JD Edwards, PeopleSoft, Siebel, or Oracle, could potentially drive up the labor costs even further.

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

[2] *See, e.g.,* December 5, 2008 Deposition of Matthew Bowden at 46:13-47:25; January 6, 2009 Deposition of Shai Agassi at 119:17-120:2; May 21, 2009 Deposition of Seth Ravin at 11:15-12:20, 19:11-21:12.

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.

IV. SCOPE OF ANALYSIS

As described in Section III above, I understand that SAP TN used copies of Oracle's PeopleSoft, JD Edwards, and Siebel enterprise software applications, as well as fixes, patches, and updates to those software applications, to provide support services to SAP TN customers. In light of the overall volume of material put at issue by SAP TN's actions, I focused the majority of my effort on a targeted subset of this material. Specifically, I analyzed the cost of development for the following Oracle products using a Function Point Analysis:

- JD Edwards EnterpriseOne, Version 8.12,
- PeopleSoft 8.8 Customer Resource Management ("CRM"),
- PeopleSoft 8.8 Human Resources Management System ("HRMS"),
- PeopleSoft 8.4 Financial Supply Chain Management - rev 1 ("FSCM"),
- PeopleSoft 8.0 Student Administration ("Student Admin"), and
- PeopleSoft 8.8 Enterprise Performance Management - rev 1 ("EPM")

These products offered the advantage of providing relatively easy access to the components of Source Code, which was required for my analysis. I also understand that these products represent the latest copyrighted versions of these products that were also supported by SAP TN.

After concluding my Function Point Analysis, I performed a COCOMO analysis on the products listed above, as well as on the JD Edwards World and Siebel products that I understand are also at issue in this litigation.

If I were to assume a less conservative posture, I could have reasonably analyzed the cumulative development costs associated with each of the prior product versions, along with the development costs associated with producing the ongoing fixes, patches, and updates for each

version. I did not do so in order to ensure that there was no double counting of any development efforts between versions. I also could have analyzed the value of time associated with acquiring instant access to the software applications, as opposed to enduring the time required for developing the cited products. Instead, however, I focused my analysis on the pure cost of development of the underlying products themselves. These examples demonstrate ways in which my report represents a conservative position.

V. FUNCTION POINT METHODOLOGY

The approach of Function Point Analysis was carefully selected, based on the existence of well-documented and widely-accepted estimating practices that provided the ability to reverse-engineer the costs associated with full life-cycle product development, using the size and complexity of the underlying Source Code as a proxy for the total cost of development.

IFPUG, www.ifpug.org, which is a non-profit, member-governed organization, provides a measurement technique called Function Point Analysis ("FPA") for the functional sizing of software. IFPUG endorses FPA as its standard methodology for software sizing. Furthermore, IFPUG participates as a Lead Member in the International Software Benchmarking Standards Group Ltd. ("ISBSG").[7]

In adopting a conservative posture in the scope of my analysis, I determined that only the most recent copyrighted versions of JD Edwards EnterpriseOne and PeopleSoft that were supported by SAP TN would be analyzed to derive the cost of development for purposes of my analysis. By focusing on the most recent versions, I thereby assumed that all of the work effort associated with creating prior versions would be accounted for when estimating the costs associated with developing the most recent version.

JD Edwards World and Siebel products are also at issue in this litigation. Although these two products were not analyzed using Function Point Analysis, the alternate method of

[7] 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]

Estimating Approach (high-level view)				
Step Number	**Input Information**	**Activity Performed**	**Technique Applied**	**Output Information**
9	Effort Allocation by Role	Apply hourly rates and determine the cost of development for each of the Analyzed Products, across a number of staffing scenarios	Apply resource costs	Estimated Cost of Development
10	Estimated Cost of Development	Analyze the estimated development costs for each of the Analyzed Products	Automated calculations	Per Unit Cost Calculations

Table 2 - Estimating Approach

VI. TEN-STEP ANALYSIS TO DETERMINE THE COST OF DEVELOPMENT USING FUNCTION POINT

A. Step One: Identify and Group Source Code Components

The purpose of identifying and grouping the Source Code components, from the total population of application components, was to identify and isolate those components from which meaningful estimates could be derived. While other components, such as application code, utilities, database files, screens, and documentation are all relevant and interesting, Function Point Analysis is designed to derive the effort required to create all of these other components as a function of understanding the size and characteristics of the associated Source Code. By applying Function Point Analysis, the development costs for all components can be extrapolated from understanding the underlying Source Code.

The entire set of software components was reviewed, with a focus on identifying the components that represented Source Code. This was done by reviewing the file extensions to identify the file types that could contain Source Code, and then opening each suspected file to confirm that it did indeed contain valid Source Code. As the components of Source Code were identified, they were then grouped by product, module, version, and programming language.[9]

[9] As the components of Source Code were identified, they were then grouped by product, module, version, and programming language. For the JD Edwards EnterpriseOne and PeopleSoft products, I was provided with the complete applications in the form of ISO files (images of CDs or hard drives), which were physically delivered in a series of external hard drives. With regard to PeopleSoft, a significant component of the Source Code was written in PeopleCode, which resided as objects within the database. For the purpose of my analysis, Oracle extracted these objects en masse from the PeopleSoft modules listed in Section IV and provided me with the set of PeopleCode as

The original input for Step One was the software products/modules for the Analyzed Products. Below, Table 3 (ORCLX-PIN-000065 Table 3) displays the number of Source Code programs, for the identified groupings.

Number of Source Code Programs					
Software Product Version	**Programming Language Groupings**	**Number of Source Code Programs**		**Totals**	
JDE EnterpriseOne Version 8.12	C	28,471	Programs	38,634	Programs
	Java J2EE	10,163	Programs		
PeopleSoft Version 8.X	COBOL/400	3,657	Programs	17,480	Programs/Files
	SQC, SQR, DMS and SQL	12,146	Programs		
	RPT and MDL	1,663	Programs		
	PeopleCode	14	Files (w/multiple programs)		
	Totals:	**56,114**	**Programs/Files**	**56,114**	**Programs/Files**

Table 3 - Source Code Programs

Detailed inventories of Source Code files, grouped by stratum, have been produced as bates number ORCLX-PIN-000063 for JD Edwards EnterpriseOne, and bates number ORCLX-PIN-000064 for PeopleSoft.

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

C. Step Three: Determine the Amount of Functionality

Step Three involves a process known as Backfiring to determine the amount of functionality. As explained above in Section V, Function Point Analysis is a method for determining the size of a software product, by describing it in terms of the amount of work being performed within the programming code. The major objective of Function Point Analysis is to describe the quantity of functionality that is contained in a component of Source Code, and to establish an objective statement of software size, which is independent of the technology in which it is written. Function Point Analysis, when paired with Backfiring, is a valuable technique for deriving the size of software in normalized terms. Backfiring refers to the process of using the end-product, in this case the Source Code, to determine the size of the application development effort that was used to produce it.

Considerable research has been performed regarding the expressive power of computer languages. In particular, this research indicates how many logical SLOCs are required to implement a Function Point of work, with a single Function Point of work consisting of an elementary process that performs one of the following types of system-related activities:[13]

[13] Function Point Counting Practices Manual, Release 4.2, ISBN 0-963-1742-9-0, The International Function Point