PROPERTY TAX VALUATION

of Computer Software

This discussion focuses on the valuation of internally developed software.

ROBERT F. REILLY, CPA

In some taxing jurisdictions,

the internally developed computer software of an industrial or commercial taxpayer company may be exempt from ad valorem state and local taxation (SALT). In these situations, the SALT property assessment should not include the value of the taxpayer's internally developed software.

In this discussion, we assume that the taxpayer is the type of company that is subject to property taxation based on the unit valuation principle. Such taxpayers often own property that cross over more than one taxing jurisdiction (like a telephone company, a railroad, or an interstate pipeline). Or, such taxpayers own property that is physically, functionally, and economically integrated (like an oil refinery, a cable TV system, or a water or wastewater system). In these instances, unless adjusted, the taxpayer's unit value conclusion typically includes the value of (1) all of the taxpayer's tangible property and (2) all of the taxpayer's intangible property.

If the taxpayer is located in a jurisdiction that taxes tangible property only, then the taxing authority should adjust the taxpayer's total unit value for the value of any intangible personal property (such as internally developed software).

This discussion focuses on the generally accepted approaches and methods that valuation analysts ("analysts") use to value internally developed software for property tax purposes. This discussion focuses on the application of the cost approach, and specifically the replacement cost new less depreciation (RCNLD) method, to value taxpayer software.

Introduction

Some taxing jurisdictions tax the commercial taxpayer's intangible personal property for ad valorem SALT purposes. Some taxing jurisdictions tax all of the tangible property and all of the intangible property of commercial taxpayers. In these taxing jurisdictions, a taxpayer's internally developed software would be subject to SALT.

However, many taxing jurisdictions only tax tangible property—that is, the real estate and/or tangible personal property—of industrial and commercial taxpayers. In these jurisdictions, the value of a taxpayer's intangible personal property (including internally developed software) would not be subject to property taxation. Taxpayers in these jurisdictions—especially industrial and commercial taxpayers subject to the unit principle of property valuation—should exclude the value of such software from the total bundle of property subject to taxation.

This discussion focuses on the valuation of internally developed software. There are generally accepted methods within the cost approach, the market approach, and the income approach methods to value software source code. This discussion focuses on the application of the cost approach, and, in particular, the RCNLD valuation method. The RCNLD method is often applied to value internally developed software source code and associated documentation and databases.

This discussion (1) describes software and (2) summarizes the cost approach RCNLD method. For the valuation of software, analysts often apply software development effort estimation models to estimate the amount of time required to replace the software. In particular, this discussion focuses on use of the COCOMO model and the SLIM model (defined below) to apply the cost approach. This discussion also presents an illustrative example of the application of the RCNLD method to value software and associated intangible property.

Definition of Software for Property Tax Purposes

Software is sometimes defined as the programs that tell the computer what to do. The broadest definition is that software includes everything that is not hardware. In Revenue Procedure 69-21, the Internal Revenue Service (the "Service") defines software as follows:

> All programs or routines used to cause a computer to perform a desired task or set of tasks and the documentation required to describe and maintain those programs. Computer programs of all classes, for example, operating systems, executive systems, monitors, compilers, and translator assembly routines, and utility programs, as well as application programs are included. "Computer Software" does not include procedures which are external to computer operations, such as instructions to transcription operators and external control procedures.[1]

Determining If the Software Is Taxable

The determination of whether software is intangible personal property is sometimes the subject of controversy in the property tax discipline. State taxing authorities have attempted to address this issue. These attempts have resulted in inconsistent state-specific rules and methods to which analysts and taxpayers look for guidance in determining what portion (if any) of software is taxable and what portion of software is not subject to taxation.

When valuing software for property tax purposes, it is important to determine whether the software is taxable or not taxable. Most taxpayer companies own and operate software that was either:

1. purchased from a seller and optimized for the taxpayer operations, or
2. internally developed by the taxpayer information technology (IT) personnel.

Some states assess property tax on internally developed software. Virginia, for example, specifically defines "computer application software" as taxable intangible personal property.[2] In general, most states do not tax intangible personal property. Taxpayers typically take the position that the source code and related documentation of the software is intangible personal property and should not be subject to property taxation.

The following three general criteria have been developed by state courts and taxing authorities to determine whether software source code is either tangible personal property or intangible personal property:

1. Whether the taxpayer purchased a tangible storage medium versus the intangible knowledge contained within.
2. Whether the software is operating (or "operational") software or application software.
3. Whether the software is internally developed or "bundled."

Criterion one. The first criterion, which is sometimes called the "container test," focuses on a substance-over-form inquiry involving two components:

1. A physical storage medium (e.g., a compact disc, digital versatile disc, or a magnetic tape).
2. The knowledge and/or information contained on the storage medium.

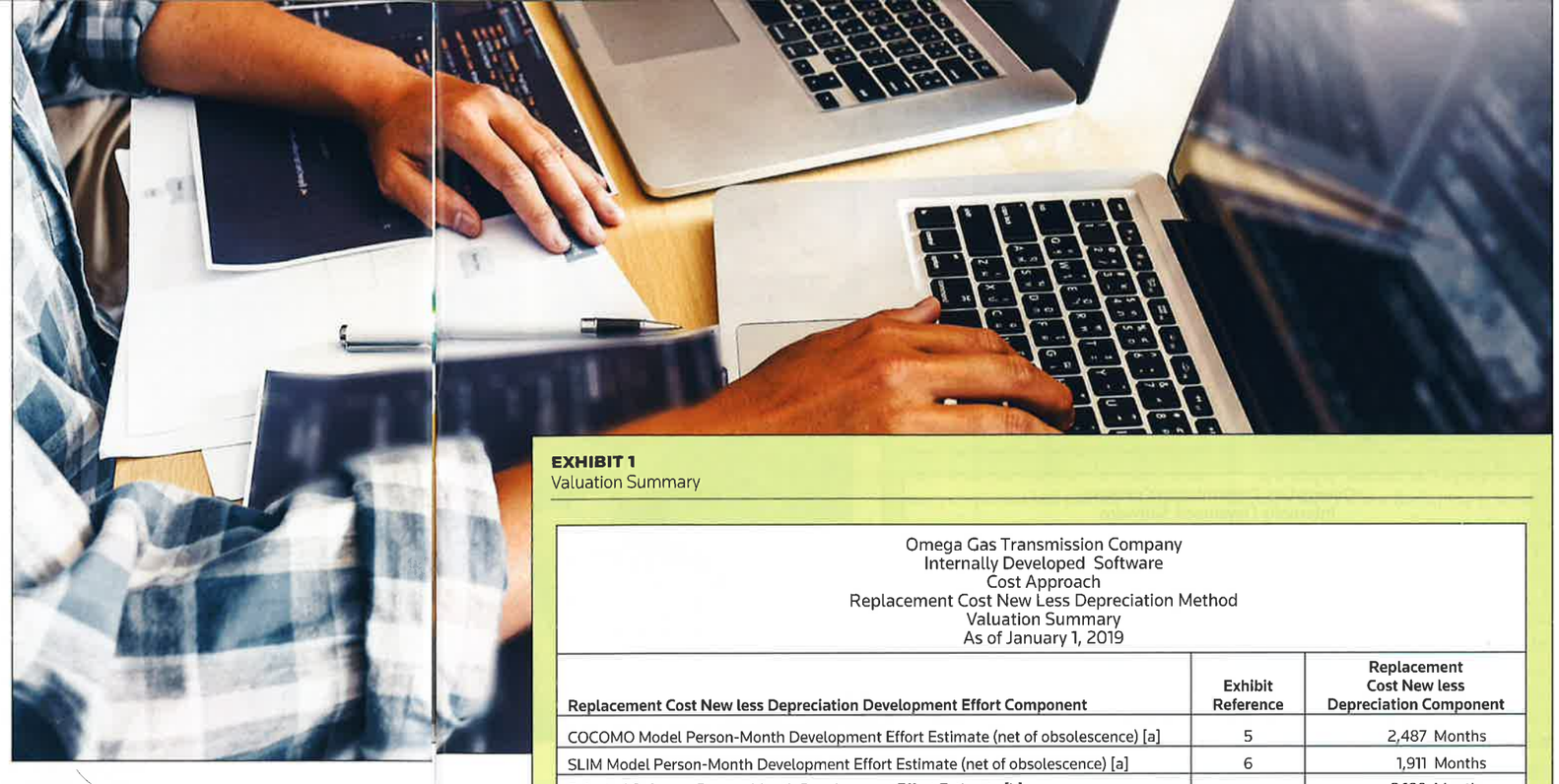


EXHIBIT 1
Valuation Summary

Omega Gas Transmission Company
Internally Developed Software
Cost Approach
Replacement Cost New Less Depreciation Method
Valuation Summary
As of January 1, 2019

Replacement Cost New less Depreciation Development Effort Component	Exhibit Reference	Replacement Cost New less Depreciation Component
COCOMO Model Person-Month Development Effort Estimate (net of obsolescence) [a]	5	2,487 Months
SLIM Model Person-Month Development Effort Estimate (net of obsolescence) [a]	6	1,911 Months
Selected Software Person-Month Development Effort Estimate [b]		2,199 Months
Software Person-Month Development Effort Estimate		2,199
Full Absorption Cost per Person-Month	2	$ 10,440
Software Replacement Cost New Indication		$ 22,954,945
Less: Additional Functional Obsolescence		$ -
Less: Economic Obsolescence		$ -
Equals: Software Value (rounded)		$ 23,000,000

[a] For purposes of this simplified illustrative example, economic obsolescence is assumed to be 0 percent.
[b] Average of COCOMO indicated person-months and SLIM indicated person-months.
Note: These data are hypothetical and are presented for illustrative purposes only.

ROBERT F. REILLY is a managing director of Willamette Management Associates in the firm's Chicago, Illinois, office and can be reached at (773) 399-4318 or rfreilly@willamette.com.

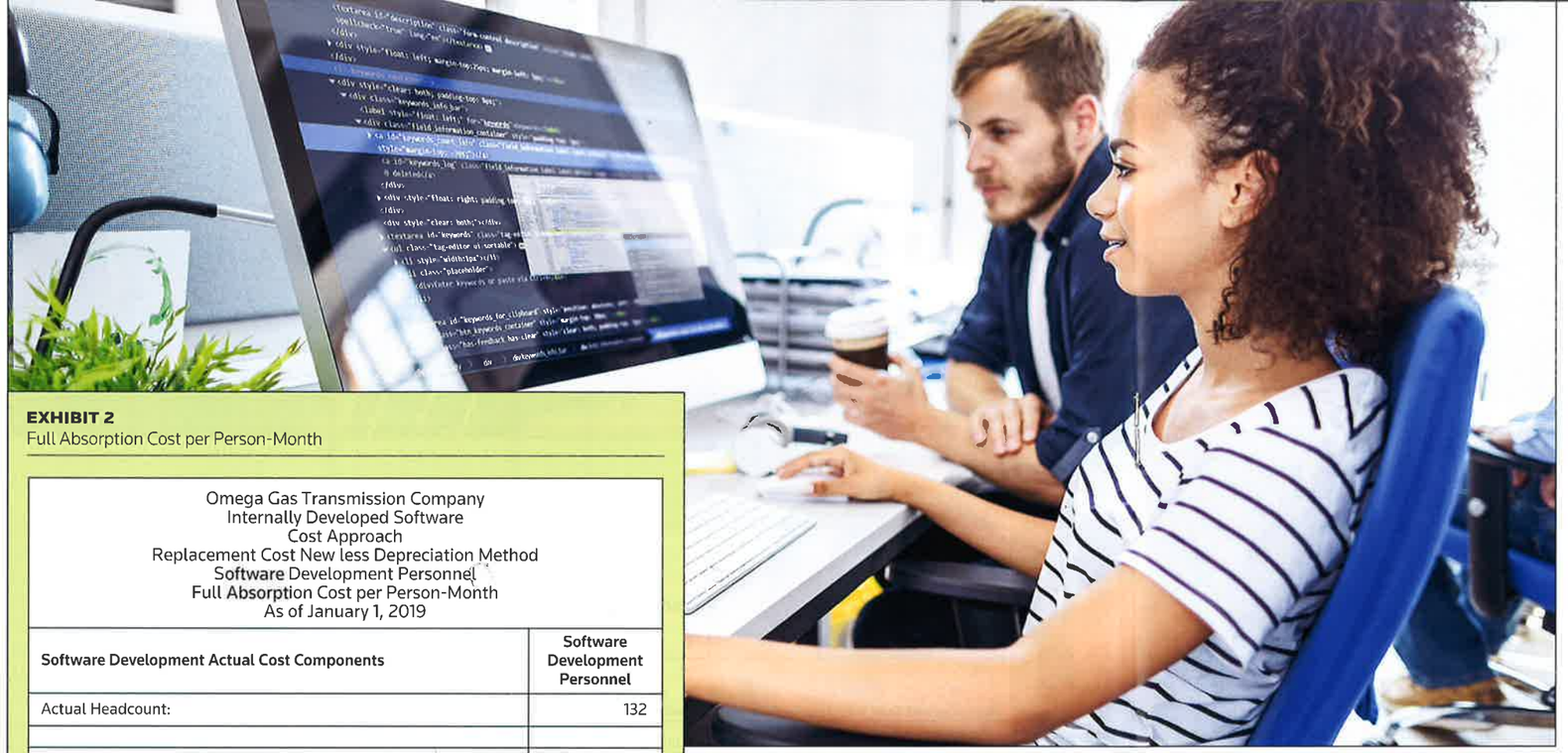


EXHIBIT 2
Full Absorption Cost per Person-Month

Omega Gas Transmission Company
Internally Developed Software
Cost Approach
Replacement Cost New less Depreciation Method
Software Development Personnel
Full Absorption Cost per Person-Month
As of January 1, 2019

Software Development Actual Cost Components	Software Development Personnel
Actual Headcount:	132
Actual Costs:	
Salaries	8,500,000
Employee Benefits	1,625,000
Bonuses	525,000
Overhead	3,412,500
Total Actual Annual Cost	14,062,500
Monthly Cost per Person:	
Total Actual Annual Cost	14,062,500
Divided by: Headcount	132
Annual Cost per Person	106,534
Divided by: 12 Months	12
Direct and Indirect Cost per Person-Month	8,878
Computer Software Developer's Profit [a]	5%
Direct Cost, Indirect Cost, and Developer's Profit per Person-Month	9,322
Direct Cost, Indirect Cost, and Developer's Profit Cost per Person-Month	9,322
Entrepreneurial Incentive as a Percent of Direct Cost, Indirect Cost, and Developer's Profit [a]	12%
Full Absorption Cost per Person-Month	10,440
Full Absorption Cost per Person-Month (rounded)	10,400

[a] Determined by the analyst (detalls not presented).
Source: Taxpayer-provided costs and headcount and analyst calculations.
Note: These data are hypothetical and are presented for illustrative purposes only.

In this context, intangible information refers to the digital manifestation of human knowledge in the form of the software code. The code instructs a microprocessor to perform computational tasks that alter and communicate this intangible information.

In the early years of computing, taxing authorities sought to characterize software by the tangible medium in which it was stored and distributed. The container test examined:

1. whether the intangible information (that is, the software code) contained within a tangible medium is a significant factor for property tax purposes, and
2. whether the tangible medium may be considered incidental to the purchase of that intangible information.

The container test is less relevant in a modern IT environment. This is because the use of a tangible storage medium for software distribution has declined. Source code is directly downloaded to computers or accessed on demand from servers in a cloud network. These methods of software distribution have made many forms of physical distribution unnecessary.

An example of the application of the container test occurred in 1996 when the Texas Court of Appeals ruled that software was considered intangible property, and, therefore, not subject to ad valorem property taxation.[3] That court concluded that the software was intangible. This was because the "essence of the transaction" was not in the tangible medium that was used to transport the software to the consumer (for example, a disk or CD-ROM) but rather the software that it contained. "Computer application software," the court reasoned, is considered intangible personal property consisting of unperceivable binary pulses, programs, routines, and symbolic mathematical code that control the function of computer hardware and direct hardware operations. Therefore, software was not subject to property taxation as tangible personal property.

Criterion two. A number of states have emphasized a second criterion that focuses on how separable the software is from the hardware on which it operates. Some states insist that software is essentially inseparable from the tangible hardware on which it operates. For example, the Ohio Supreme Court upheld the Ohio Department of Taxation position that all software was subject to property taxation under the reasoning that the coded instructions are always stored in some form of physical memory—a tangible medium—when operating in a computer.[4] Therefore, in Ohio, all internally developed software may be subject to property taxation.

In other states, the issue of the ability to separate software from the computer usually takes the form of classifying software as either:

1. operating software or
2. application software.

Operating software is generally required in order for the computer to function properly. Sometimes operating software is described as "embedded" software or "firmware." This label is based on the fact that the software is coded into memory chips attached directly to the circuit board of a computing device.

A laptop computer contains embedded software in the form of a basic input output system (BIOS). A BIOS is permanently stored in a memory chip on a computer motherboard (the primary circuit board). It is automatically executed when the computer is turned on. The BIOS serves as the fundamental operating system (OS) for managing the microprocessor(s) on the motherboard and the peripheral devices that attach to the motherboard. For a laptop computer, these attached devices may include a hard drive, a video graphics card, a keyboard, and a touchpad.

Depending on the taxing jurisdiction, operating software may have a more expansive definition. That more expansive definition includes a general-purpose OS that works in conjunction with the BIOS.

The Kansas Department of Revenue described the distinction between operating software and application software as follows:

> The Kansas Supreme Court has held that software programs are taxable if they are operational programs; programs the computer cannot operate without. These programs are considered an essential portion of the computer hardware and are taxable as tangible personal property in conjunction with the hardware. On the other hand, application programs, which are particularized instructions, are intangible property, which is not subject to taxation in Kansas.[5]

The California State Board of Equalization has concluded the following with regard to criterion two:

> In general, software is classified as nontaxable property. The one exception to this general rule is software that is considered a "basic operational program" or "control program." These terms refer to *a computer program that is fundamental and necessary to the functioning of a computer.* All other software (sometimes called *application software*) is nontaxable. But if the application software comes bundled with the computer hardware or other equipment at a single price and the taxpayer does not provide the assessor with information that will enable the assessor to separately estimate its value, then the assessor may consider the total bundled price as indicative of the value of the taxable tangible property.[6]

As a simple illustration, a laptop first executes a BIOS when the laptop is turned on. In some taxing jurisdictions, this BIOS may be considered tangible personal property that is subject to property taxation. Once the laptop has started operating, a user may decide to execute an application such as Microsoft Office. Microsoft Office may qualify as tax-exempt application software. This is because such software executes "on top" of the BIOS, and it is not required for the computer to operate. That is, the laptop will function normally regardless of whether Microsoft Office is installed. The classification of the Windows OS, which also executes on top of the BIOS, as taxable operating software or as nontaxable application software may vary by taxing jurisdiction.

This interplay of embedded operational software and general purpose operating systems may lead to complicated tax rules. The operating software/application software dichotomy only offers a general guideline. However, not all operating software is subject to property tax and not all application software is tax exempt.

Criterion three. The third criterion classifies software as either:

1. software that is developed for internal use or
2. software that is developed for commercialization (that is, for resale)—"bundled" software.

Bundled software typically includes software that is licensed to others and may be held by the developer as inventory. Under some state property tax statutes, internally developed software is taxed, while bundled software is not. An example of bundled software is the Microsoft Office software suite. If a taxpayer company purchases Microsoft Office along

with a new laptop computer, the value of Microsoft Office ordinarily would not be included in the tax base (let's assume that the subject taxing jurisdiction excludes bundled software), while the value of the laptop itself would be included as tangible property.

This concept is fairly consistent with the operational software/application software distinction. The distinction in this criterion becomes more evident if the analyst considers that the taxpayer may be taxed on its laptop software if it instead internally develops an application with word processing and other office productivity features. Under this criterion, taxability depends on the issue of customization, not on whether the software is application software.

In practice, discerning between internally developed software and bundled software may be difficult. It may be difficult to determine the taxability of the software when the analyst considers the many ways in which software can be created, modified, and distributed. If a software developer creates software for a particular customer's needs that will not be resold to others, it may be considered internally developed software. However, if the developer creates the same software for a chain of franchise businesses and then licenses the software individually to 100 franchisees, some taxing jurisdictions may classify the software as having been developed for commercialization. This result may occur even though the customers belong to the same franchise chain.

Software Valuation Approaches and Methods

As summarized below, there are three generally accepted approaches to intangible personal property valuation:

1. Cost Approach—The cost approach estimates the value of intangible property as the cost (in terms of current dollar expenditures) required to create a property with equivalent utility and functionality as the actual property. Analysts typically consider the following cost components in a cost approach analysis: direct costs, indirect costs, developer's profit, and entrepreneurial incentive.

If the replacement property is superior to the actual property, then allowances may be made for the various forms of obsolescence, including functional (including technological) obsolescence and external (including economic) obsolescence.

2. Market Approach—The market approach estimates the value of intangible property based on valuation pricing multiples derived from the market-based arm's-length sale or license transactions involving either comparable or guideline intangible property. Typically, individual intangible property is not bought and sold in fee simple interest. Accordingly, intangible property sale transactional data are not often readily available.

However, many intangible properties (such as trademarks, copyrights, and patents) are licensed in arm's-length transactions. When available, these transactional data may be used to prepare a market approach analysis.

3. Income Approach—The income approach recognizes the prospective revenue, expense, profitability, and investments associated with the ownership of intangible property. This approach estimates the value of an intangible property as the present value of future income. That income may be defined as operating income, net income, net cash flow, operating cash flow, or some other measure of income, and that income may be estimated over the property's expected useful economic life (UEL).

This expected income stream is brought to a present value by the use of an appropriate market-derived, risk-adjusted rate of return.

This discussion focuses on the application of the cost approach, and specifically the RCNLD method.

Cost Approach

The cost approach concludes a value for the software based on some measure of cost. The cost measures that are typically applied in the cost approach include the following:

1. Reproduction cost new (RPCN)
2. Replacement cost new (RCN)

The RPCN reflects the cost to create an exact replica of the taxpayer's software. The RPCN reflects the cost to create the functionality or utility of the software, in a form that is identical to the actual software. Functionality refers to the ability of the taxpayer software to perform the task for which it was designed. Utility refers to the ability of the taxpayer software to provide an equivalent amount of satisfaction to the user or beneficiary of the actual software.

The RCN refers to the cost to create the functionality or utility of the taxpayer software, but in a form or appearance that may be quite different from the taxpayer's actual software. While the replacement software performs the same task as the taxpayer software, the replacement software is often superior (in some way) to the actual software. That is, the replacement software may yield more satisfaction. If this is the case, the analyst may adjust for this factor in an obsolescence estimation. Adjustments for obsolescence are discussed below.

The methods that analysts may use to estimate the software RPCN or RCN include the following:

1. The trended historical cost
2. The software engineering development effort estimation models

Trended historical cost. In this method, actual historical software development costs are identified and quantified. These actual costs are then "trended" through the valuation date by an appropriate inflation-based index factor. The analyst ordinarily includes all costs associated with the software development.

An allocation of taxpayer overhead costs and the cost of employee fringe benefits ordinarily may be included in addition to employee payroll costs if the taxpayer personnel are employed in tasks related to the software development.

Historical costs ordinarily should include (or should be adjusted to include) an allowance for the developer's profit on the software development project, an al-

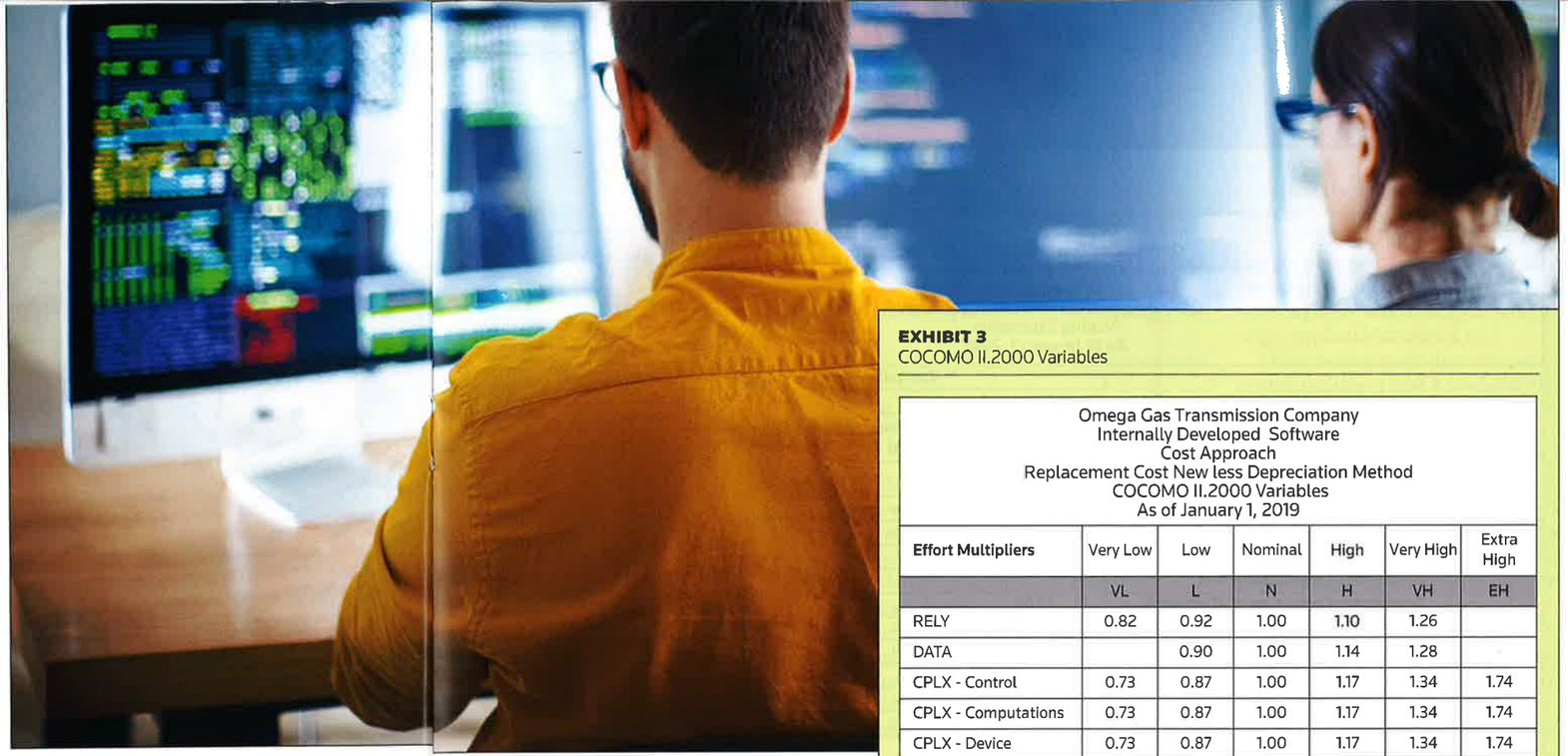


EXHIBIT 3
COCOMO II.2000 Variables

Omega Gas Transmission Company
Internally Developed Software
Cost Approach
Replacement Cost New less Depreciation Method
COCOMO II.2000 Variables
As of January 1, 2019

Effort Multipliers	Very Low	Low	Nominal	High	Very High	Extra High
	VL	L	N	H	VH	EH
RELY	0.82	0.92	1.00	1.10	1.26	
DATA		0.90	1.00	1.14	1.28	
CPLX - Control	0.73	0.87	1.00	1.17	1.34	1.74
CPLX - Computations	0.73	0.87	1.00	1.17	1.34	1.74
CPLX - Device	0.73	0.87	1.00	1.17	1.34	1.74
CPLX - Data	0.73	0.87	1.00	1.17	1.34	1.74
CPLX - User	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	
TIME			1.00	1.11	1.29	1.63
STOR			1.00	1.05	1.17	1.46
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.42	1.19	1.00	0.85	0.71	
PCAP	1.34	1.15	1.00	0.88	0.76	
PCON	1.29	1.12	1.00	0.90	0.81	
AEXP	1.22	1.10	1.00	0.88	0.81	
PEXP	1.19	1.09	1.00	0.91	0.85	
LTEX	1.20	1.09	1.00	0.91	0.84	
TOOL	1.17	1.09	1.00	0.90	0.78	
SITE - Collocation	1.22	1.09	1.00	0.93	0.86	0.80
SITE - Communications	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	
Scaling Factors:						
	VL	L	N	H	VH	EH
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Note: These data are hypothetical and are presented for illustrative purposes only.

1 Rev. Proc. 69-21, 1969-2 C.B. 303.

2 Rulings of the Tax Commissioner, Virginia Department of Taxation, Document 13-47 (April 4, 2013).

3 *See Dallas Cent. Appraisal Dist. v. Tech Data*, 930 S.W.2d 119 (Tex. App. 1996).

4 *See Andrew Jergens Co. v. Tax Comm'r*, 848 N.E.2d 499 (Ohio 2006).

5 *See* https://law.justia.com/cases/kansas/supreme-court/1986/58-619-1.html.

6 *See* https://www.boe.ca.gov/proptaxes/embedded_software.htm.

7 Capers Jones, *Estimating Software Costs: Bringing Realism to Estimating*, 2nd ed. (New York: McGraw-Hill, 2007), 8.

8 *Id.*, 9.

9 For a detailed description of COCOMO, see Barry W. Boehm, *Software Engineering Economics* (New York: Prentice-Hall, 1981).

10 For a detailed description of COCOMO II, *see* Boehm, et al., *Software Cost Estimation with COCOMO II* (New York: Prentice-Hall PTR, 2000).

11 *See* http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.

12 Robert F. Reilly and Robert P. Schweihs, *Guide to Intangible Asset Valuation* (New York: American Institute of Certified Public Accountants, 2013), 229.

EXHIBIT 4
Cost Driver Ratings

Omega Gas Transmission Company
Internally Developed Software
Cost Approach
COCOMO II.2000 Effort Multipliers and
Scaling Exponents
As of January 1, 2019

Software Development Cost Drivers			Computer Software Programs								
			Program 1			Program 2			Program 3		
			Rating [a]		Effort Multiplier	Rating [a]		Effort Multiplier	Rating [a]		Effort Multiplier
PRODUCT FACTORS											
	RELY	Required System Reliability	L		0.92	N		1.00	H		1.00
	DATA	Data Base Size	N		1.00	N		1.00	N		1.00
	CPLX	Software System Complexity			0.89			0.92			0.92
		Complexity - Control Operations	N			L	0.87		VL	0.73	
		Complexity - Computational Operations	VL	1.00		L	0.87		N	1.00	
		Complexity - Device-Dependent Operations	N	0.73		N	1.00		L	0.87	
		Complexity - Data Management Operations	N	1.00		N	1.00		N	1.00	
		Complexity - User Interface	VL	1.00		L	0.87		N	1.00	
	RUSE	Required Reusability	N	0.73	1.00	N		1.00	N		1.00
	DOCU	Documentation Match to Life-Cycle Needs	N		1.00	VL		0.81	N		1.00
COMPUTER FACTORS											
	TIME	Execution Time Constraint	N		1.00	N		1.00	N		1.00
	STOR	Main Storage Constraint	N		1.00	N		1.00	N		1.00
	PVOL	Platform Volatility	L		0.87	N		1.00	L		0.87
PERSONNEL FACTORS											
	ACAP	Analyst Capability	N		1.00	VH		0.71	N		1.00
	PCAP	Programmer Capability	VH		0.76	H		0.88	H		0.88
	PCON	Personnel Continuity	N		1.00	N		1.00	VH		0.81
	AEXP	Applications Experience	VH		0.81	H		0.88	H		0.88
	PEXP	Platform Experience	H		0.91	N		1.00	H		0.91
	LTEX	Language and Tool Experience	N		1.00	N		1.00	N		1.00
PROJECT FACTORS											
	TOOL	Use of Software Tools	VH		0.78	N		1.00	N		1.00
	SITE	Multisite Development			0.80			1.00			1.11
		Site Collocation	EH	0.80		N	1.00		N	1.00	
		Communications Support	EH	0.80		N	1.00		VL	1.22	
	SCED	Required Development Schedule	H		1.00	N		1.00	N		1.00
		Product of the Effort Multipliers			0.25			0.41			0.56
		Scale Drivers	Rating		Scale Factor	Rating		Scale Factor	Rating		Scale Factor
SCALE FACTORS											
	PREC	Precedentedness	H		2.48	VH		1.24	N		3.72
	FLEX	Development Flexibility	H		2.03	N		3.04	H		2.03
	RESL	Architecture/Risk Resolution	H		2.83	N		4.24	N		4.24
	TEAM	Team Cohesion	N		3.29	N		3.29	L		4.38
	PMAT	Process Maturity	N		4.68	N		4.68	N		4.68
		Sum of the Scale Factors			15.31			16.49			19.05
		Scaling Exponent			1.0631			1.0749			1.1005

[a] Provided by Omega software development personnel.
Note: These data are hypothetical and are presented for illustrative purposes only.

lowance for entrepreneurial incentive to motivate the development project, all direct development costs such as salaries and wages, and all indirect development costs, such as taxpayer company overhead and employment taxes/employee benefits.

The application of the trended historical cost method typically estimates the software RPCN. In many cases, due to technological advances in programming languages or programming tools, for example, the software RCN may be lower than the software RPCN.

Software engineering development effort estimation models. The analyst may apply development effort estimation models to estimate either the software RPCN or RCN. Generally, development effort measurement models were originally developed to assist software engineers in estimating the amount of effort, time, and human resources needed to complete a software project. These models have been adapted for valuation purposes.

The primary input to the software engineering cost estimation models is a size-related metric. Capers Jones, an authority on software cost estimation, observed: "Every form of estimation and every commercial software cost-estimating tool needs the sizes of key deliverables in order to complete an estimate."[7] Jones lists six types of sizing:

1. Sizing based on lines of code
2. Sizing by extrapolation from function point analysis
3. Sizing by analogy with similar products of known size
4. Sizing based on "project manager's intuition"
5. Sizing based on "programmer's intuition"
6. Sizing using statistical methods or Monte Carlo simulation[8]

One sizing metric is the number of lines of code. The definition of a line of code and the associated line of code counting conventions vary among the development effort estimation models. One definition of a line of code is as source code instructions (i.e., instructions as written by software engineers) or object code instructions (what the computer produces after it has compiled, or translated, the source code into instructions the computer can more directly process).

Lines of code have meaning only within the context of the software language employed. Languages have evolved over time and can be classified into generations. As a general observation, higher-generation languages (i.e., more modern programming languages) require less source code to perform the same tasks than lower-generation languages.

The software valuation can also be developed using different base size units than source lines of code. Examples of these include both function points and object points.

The software engineering development effort estimation models include the following:

1. The Constructive Cost Model (COCOMO) and its derivatives
2. The Software Lifecycle Management (SLIM) model

These software engineering development effort estimation models are considered "algorithmic" models. This is because they generate effort estimates using a set of quantified inputs, such as number of lines of source code, which is processed automatically in accordance with metrics and formulas derived from the empirical analysis of large databases of actual software projects.

Typically, the development effort estimation models calculate an estimate of the amount of effort required to develop the software, expressed in terms of person-months. The number of person-months is multiplied by a blended cost per person-month to arrive at a cost measurement for the software. The blended cost per person-month is typically a full absorption cost (e.g., the cost of a software engineer would include benefits, wages, applicable overhead, etc.).

Other development effort estimation models include (1) the KnowledgePlan (kPLAN) model and (2) the SEER for Software ("SEER-SEM") model.

kPLAN. The kPLAN model is a function point-driven model that incorporates a historical knowledge database of project data derived from over 11,000 software projects that have been collected and researched by Software Productivity Research, LLC (SPR).

The specific algorithms applied by kPLAN are proprietary. The model uses functional metrics to derive predictive/analytical productivity rates given a significant number of known (or assumed) parameters. Projects are classified by, among other things, scope (e.g., program or application, subsystem), topology (e.g., stand alone, client/server), class (e.g., end-user developed, IT developed), and type (e.g., interactive graphical user interface, multimedia).

The size of the subject software system can be expressed in multiple ways, including function points or lines of code, by language. The analyst assigns attribute values that describe the personnel, technology, process, environment, and product. kPLAN was updated in 2011 with the release of version 4.4. However, SPR ceased support for the development effort estimation model. The model is still available for download from various software archive websites.

SEER-SEM. The SEER-SEM model is an algorithmic project management tool designed to estimate, plan, and monitor the estimated effort and resources necessary for software development projects. SEER-SEM is actually a group of models working in concert to provide estimates of software development effort, duration, staffing, and defects.

The following list presents the specific SEER-SEM metrics and the questions that these metrics address:

1. Sizing (how large is the project?)
2. Technology (how productive are the developers?)

3. Effort and Schedule Calculation (what amount of effort and time is needed?)
4. Constrained Effort/Schedule Calculation (how does the expected outcome change with constraints?)
5. Activity and Labor Allocation (how should tasks and labor be allocated?)
6. Cost Calculation (given the effort, duration, and labor, how much will the project cost?)
7. Defect Calculation (what is the expected quality of the delivered software?)
8. Maintenance Effort Calculation (how much maintenance will be required?)
9. Progress (how is the project progressing and is it on track to target completion?)
10. Validity (is the project feasible based on the technology involved?)

The current SEER-SEM version (version 7.3) is the first version of the model to incorporate all stages of the software project life cycle. This version relies on parametric modeling that applies a database of over 20,000 historical software projects to estimate required project effort and resources.

This discussion focuses on the application of the COCOMO model and the SLIM model.

COCOMO. The first generation of COCOMO was developed in the 1980s. COCOMO was developed by Barry Boehm, and is described in *Software Engineering Economics*.[9] This development effort estimation model projects the amount of effort required to develop the subject software, taking into consideration the size of the programs, the program characteristics, and the environment in which they are to be developed.

Boehm defined an effort equation in the basic COCOMO model that estimates the number of person-months required to develop a software product as a function of delivered source instructions. This person-month estimate includes all phases of the development from product design through integration and testing, including documentation.

Delivered source instructions include job control language, format statements, and data definitions. These delivered source instructions do not include comments. The basic COCOMO model allows for three different software development modes, with a specific effort equation provided for each development mode.

Boehm also introduced the intermediate COCOMO model, which refined the basic COCOMO model by introducing 15 cost drivers with associated effort multipliers. The product of these multipliers is defined as the effort adjustment factor.

The intermediate COCOMO model modified the three effort equations of the basic COCOMO model by:

1. adjusting the coefficients in the equations and
2. including the effort adjustment factor as a variable in the equations.

A more updated model, COCOMO II, was developed by researchers at the University of Southern California (USC).[10] The updated model supports the effort estimation of a variety of third and fourth generation language-based projects. It also incorporates function point analysis as well as adds two new effort drivers. An online estimation tool encompassing the COCOMO II model is available through the USC Center for Systems and Software engineering website.[11]

COCOMO II actually consists of three separate models. The most recent and detailed of the three models is the COCOMO II.2000 post-architecture model. The post-architecture model allows for increased effort due to breakage (i.e., code thrown away due to volatility in project requirements) and for automatically translated and adapted lines of code. Later, this discussion provides an illustrative example of a cost approach valuation that applies COCOMO II.

The COCOMO II model post-architecture software development equation is defined as follows:

$PM = A \times (KNSLOC)E \times \pi EM$

where:

PM = Person-months of estimated effort

A = 2.94, the effort coefficient

KNSLOC = Thousands of new source lines of code

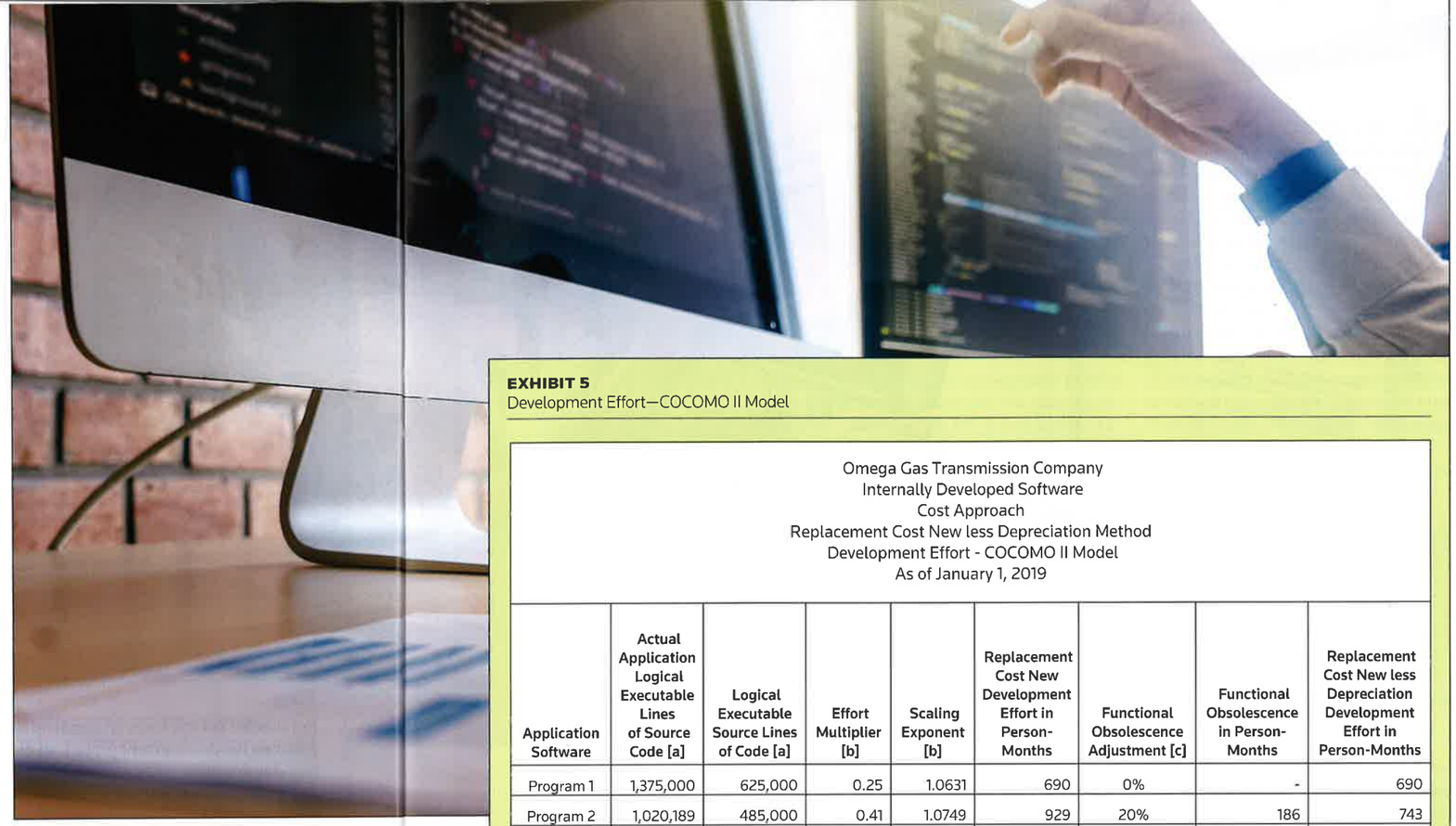
E = The scaling exponent for effort, a function of the scale factors

π EM = The product of the 17 effort multipliers associated with the cost drivers

The scaling exponent E is defined as follows:

$E = B + (0.01 \times \Sigma SF)$

where:



B = 0.91, the scaling base-exponent for effort

Σ SF = The sum of the five scale factors

A third model, COCOMO III, is currently being developed by USC and its project partners with the aim of improving the model with new and updated software cost drivers and new development paradigms.

SLIM. The SLIM development effort model was developed by Quantitative Software Management, Inc. (QSM). QSM licenses various software development effort estimation tools incorporating the model. The SLIM model (also referred to by commentators and in the academic literature as the "Putnam model") estimates the amount of effort in person-months required to develop software based on the following parameters:

1. A project size build-up parameter (a number representing a range from entirely new software to rebuilt software)
2. The software delivery time
3. The effort required to create the software
4. The expected rate of defective software
5. A productivity environment factor

The SLIM model applies a knowledge base of project data derived from over 13,000 actual software projects that were collected and researched by QSM. The SLIM model is regularly updated in order to provide accurate estimates as information technology improves.

The SLIM model allows users to specify the subject software project's environment by identifying the industry function for which that software will be used. The SLIM model applies a primary trend group to benchmark the software against the QSM industry database and compares software development projects.

The QSM primary trend groups include (1) all systems, (2) microcode and firmware, (3) real time, (4) system software, (5) command and control, (6) telecommunications, (7) scientific, (8) process control, (9) business, (10) real time, (11) engineering, (12) business agile, (13) business financial, (14) business government, (15) business web, and (16) package implementation.

The SLIM model also allows users to alter their software development estimates based on various sizing units. The base size unit is source lines of code. Below, this discussion presents an illustrative development effort estimation analysis that applies the SLIM model.

Source lines of code adjustments. The software engineering development effort estimation models often rely on an input of source lines of code. The analyst may need to make adjustments to the taxpayer-provided source lines of code. These line of code adjustments may include:

1. removing copybook lines of code,
2. determining any differences between "actual" and "ideal" source lines of code, and
3. adjusting physical source lines of code to reflect logical executable lines of source code.

Copybook lines of code. In an effort to reduce the amount of time required to write large quantities of code, software

EXHIBIT 5
Development Effort—COCOMO II Model

Omega Gas Transmission Company
Internally Developed Software
Cost Approach
Replacement Cost New less Depreciation Method
Development Effort - COCOMO II Model
As of January 1, 2019

Application Software	Actual Application Logical Executable Lines of Source Code [a]	Logical Executable Source Lines of Code [a]	Effort Multiplier [b]	Scaling Exponent [b]	Replacement Cost New Development Effort in Person-Months	Functional Obsolescence Adjustment [c]	Functional Obsolescence in Person-Months	Replacement Cost New less Depreciation Development Effort in Person-Months
Program 1	1,375,000	625,000	0.25	1.0631	690	0%	-	690
Program 2	1,020,189	485,000	0.41	1.0749	929	20%	186	743
Program 3	35,000	355,000	0.56	1.1005	1,055	0%	-	1,055
	2,430,189	1,465,000			2,674		186	2,488

[a] Omega management provided the logical executable source lines of code for the software.
[b] As presented in Exhibit 4.
[c] A 20 percent obsolescence adjustment was applied for program 2 based on eight years remaining of a 10-year UEL of the program, as indicated by Omega IT personnel.
Note: These data are hypothetical and are presented for illustrative purposes only.

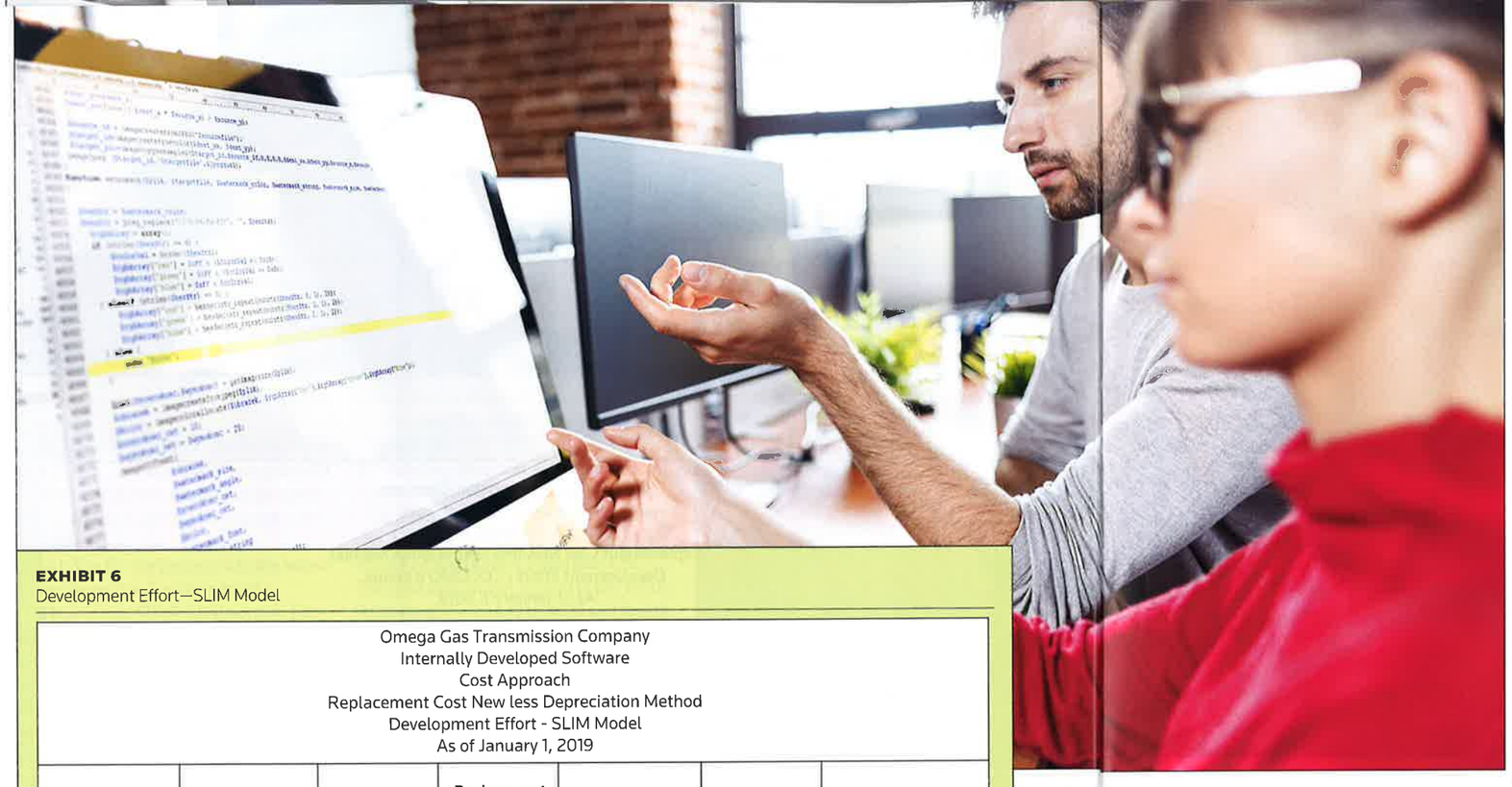


EXHIBIT 6
Development Effort—SLIM Model

Omega Gas Transmission Company
Internally Developed Software
Cost Approach
Replacement Cost New less Depreciation Method
Development Effort - SLIM Model
As of January 1, 2019

Application Software	Primary Trend Group [a]	Logical Executable Source Lines of Code [b]	Replacement Cost New Development Effort in Person-Months [c]	Functional Obsolescence Adjustment [d]	Functional Obsolescence in Person-Months	Replacement Cost New less Depreciation Development Effort in Person-Months
Program 1	Business	625,000	820	0%	-	820
Program 2	Business	485,000	684	20%	137	547
Program 3	Business	355,000	544	0%	-	544
		1,465,000	2,048		137	1,911

[a] Based on the planned use and function of the software programs.
[b] Omega management provided the logical executable source lines of code for the software.
[c] Derived by the analyst applying the SLIM software engineering cost estimation model (details not presented).
[d] A 20 percent obsolescence adjustment was applied for program 2 based on eight years remaining of a 10-year UEL of the program, as indicated by Omega IT personnel.
Note: These data are hypothetical and are presented for illustrative purposes only.

developers may use copybooks as a way to limit the amount of duplicate code that needs to be written for a particular program. Copybooks may be written once and then copied into the source lines of code for multiple programs. If the analyst included all copybooks found in any internally developed software, the number of source lines of code may be overstated.

The analyst may make an effort to determine how many copybook lines of code are original (i.e., written) and how many copybook lines of code are duplicative (i.e., copied). The analyst may reduce the source lines of code to include only the originally written copybook lines of code.

Actual and ideal source lines of code. The analyst may encounter software that would not be written in the same language if replaced or may simply be written more efficiently if replaced. These cases may be classified as "actual" and "ideal" lines of code. The adjustment for differences between "actual" and "ideal" source lines of code may be a result of individual software developer style or differences in the programming language used.

When performing an RCN analysis, the analyst may determine which, if any, programs would be written in a higher-generation language (which tends to be more efficient and requires less written code) and whether or not those programs would be replaced using fewer source lines of code.

Physical executable to logical executable lines of code. The specific line of code size measure that is used by both COCOMO II and SLIM is logical executable lines of code. In order to define logical executable lines of code, the analyst should understand:

1. the difference between logical and physical lines of code, and
2. the difference between executable and nonexecutable lines of code.

A physical line of code may be thought of as:

1. one line as typed by a programmer (i.e., before deliberately beginning a new line), or
2. one printed line on a program listing.

A logical line of code can be thought of as one logical program instruction. Many programming languages allow the software engineer to spread one logical program instruction over two or more physical lines. Some programming languages allow the software engineer to place two or more logical program instructions on the same physical line. Therefore, the number of logical lines of code in a program is generally less than the number of physical lines of code in that program.

Executable lines of code are those lines of code that are ultimately executed when the program is run (though the source lines of code will first be converted to machine code). Examples of nonexecutable lines of code are comment lines and blank lines. In other words, the program would run in the same manner regardless of the number of comment lines and blank lines. The use of logical executable lines of code reduces the effect of programmer style on the number of source lines of code, focusing instead on the functionality of the source lines of code. If necessary, the analyst may adjust physical lines of code to reflect logical executable lines of code.

Obsolescence Adjustments

When applying the cost approach to value software, the analyst should make necessary adjustments for obsolescence. Adjustments are made to account for losses in value resulting from:

1. physical deterioration,
2. functional obsolescence, and
3. external obsolescence.

Physical deterioration is a loss in value of the intangible property brought about by wear and tear, action of the elements, disintegration, use in service, and all physical factors that may reduce life and serviceability.

Functional obsolescence is the loss in value caused by the inability of the intangible property to adequately perform the function for which it is utilized. Functional obsolescence is, therefore, internal to the intangible property. Functional obsolescence is often related to such factors as property superadequacies, excess property operating costs, and property inadequacies.

External obsolescence (and economic obsolescence, a component of external obsolescence) is a loss in value caused by external forces, such as changes in the supply/demand relationship, legislative enactments, and other external factors. Those other external factors may include industry and local economic conditions that affect the value of the intangible property.

In a software valuation, all forms of obsolescence may be considered. Functional obsolescence may not be evident in software that is properly maintained. However, the analyst may consider the extent of any functional obsolescence.

When an RPCN method, such as the trended historical cost method, is applied to value software, technological obsolescence can be significant. This factor is due to increasing productivity and technological advances over time.

The application of an RCN method typically eliminates the productivity-related technological obsolescence. However, other adjustments for technological obsolescence may be necessary. The economic obsolescence component of external obsolescence usually has more relevance with respect to product software. This form of obsolescence may be examined in the valuation of operational software as well.

Although the depreciation of tangible personal property is often estimated using depreciation schedules, properly maintained software typically does not become obsolete in any predictable, continuous way.

Software value tends to vary over time by a relatively small amount due to (1) increasing productivity/technological advances, on the one hand, and (2) increasing labor costs and software enhancements, on the other hand, until the (usually unpredictable) point in time that its replacement is contemplated, for any number of reasons. Therefore, the estimate of obsolescence for properly maintained software by "depreciating" it over some time period may be unsupportable.

Useful economic life analysis. The useful economic life (UEL) estimation may be a consideration in each of the software valuation approaches. In the cost approach, a UEL analysis may be performed in order to estimate the total amount of obsolescence, if any, from the estimated measure of cost—that is, either RPCN, RCN, or trended historical cost. The analyst's assessment of UEL may have a measurable effect on the software value. Normally, a longer UEL would indicate a higher software value. A shorter UEL would indicate a lower software value.

Cost per person-time. The cost per person-time (where time is measured in hours, months, or years) is a full absorption cost. That cost includes the average base salary of the software development team and other factors. These other factors include, but are not limited to, perquisites, payroll taxes, employee benefits (life, health, disability, and dental insurance, pension plans, and continuing education), and an allocation of overhead (which includes secretarial support, office space, computer use, supplies, marketing, management, and supervisory time).

The analyst may gather information regarding the number of software devel-

opment employees, their job grades or level, as well as job titles within the IT department, and the average salary by job title. The analyst may also require data regarding the various overhead factors, such as retirement plans, medical and life insurance, company pension plan contribution, and salary incentives and bonuses.

The analyst may also have to make adjustments for (1) developer's profit and (2) entrepreneurial incentive into the full absorption cost estimate. A discussion of these cost components follows.

Developer's profit. Developer's profit is the expected return an intangible asset developer expects to receive over the direct costs and indirect costs (including materials, labor, and overhead) related to the asset development.[12] The analyst may estimate the developer's profit as a percentage return on the taxpayer's investment in direct costs and indirect costs to replace the software.

The analyst may consider guideline publicly traded companies in the computer programming services industry in order to identify a reasonable developer's profit. One procedure is to analyze the operating profit margins of a selection of guideline publicly traded companies. Since the operating profit margin is based on a return on sales and the developer's profit is based on the cost of development, the analyst may convert the selected operating profit margin to a developer's profit margin applying the following formula:

Operating profit margin ÷ (1 – Operating profit margin) = Developer's profit margin

The developer's profit margin that results from this formula is a percentage that is applied to the direct cost and indirect cost of development to calculate the total direct cost, indirect cost, and developer's profit. For example, operating profit that is 7.7 percent greater than the total cost of development is mathematically equivalent to a profit margin of 7.1 percent (minor differences are due to rounding). If a developer incurred total direct and indirect development costs of $100.00, the developer would require income of $107.70 (i.e., $7.70 of profit) to achieve an operating profit margin of 7.1 percent. In this example, the operating profit margin is calculated as $7.70 of profit divided by $107.70 of total income.

Entrepreneurial incentive. The analyst may also estimate entrepreneurial incentive by considering the following:

1. A rate of return, as indicated by the taxpayer management.
2. The estimate of the amount of time required to replace the software, as indicated by the taxpayer management.
3. The sum of the estimated software developer's profit and direct and indirect replacement costs incurred during the estimated time required to replace the software.

Entrepreneurial incentive may consider estimates of the amount of time required to replace the software.

Illustrative Software Valuation Example

Let's assume that Omega Gas Transmission Company ("Omega") is an intrastate natural gas pipeline company. Omega is assessed in its taxing jurisdiction based on the unit valuation principle. Let's assume that the assessment authority values the Omega total unit of operating property at $100 million as of January 1, 2019. Let's assume that intangible personal property is not subject to property taxation in the taxing jurisdiction. Omega owns internally developed software that is used to operate its compressor stations and its pipeline operations.

Omega retained an analyst to estimate the value of this software as of January 1, 2019, the relevant SALT valuation date. Based on this valuation, the taxpayer can exclude the value of that intangible personal property from the total unit value.

The analyst decided to apply the cost approach and the RCNLD method to value the Omega software as of January 1, 2019. To simplify this illustrative example, let's assume that software is the only intangible personal property that is owned and operated by Omega as of January 1, 2019.

Summary of exhibits. Exhibit 1 presents the summary of the RCNLD value indications using several software engineering development effort estimation models.

Exhibit 2 presents the full absorption cost per person-month used in the valuation of the Omega software. This analysis includes direct costs and indirect costs, as well as the developer's profit and entrepreneurial incentive.

Exhibit 3 presents the effort multiplier and scaling exponent factors used in the COCOMO II software development effort estimation formula.

Exhibit 4 presents the cost driver ratings and associated effort multipliers and scaling exponent factors attributable to the subject software.

Exhibit 5 presents the application of the COCOMO II model in determining the person-months required to replace the subject software.

Exhibit 6 presents the application of the SLIM model in determining the person-months required to replace the subject software.

Cost approach—RCNLD method. The process of how the analyst performs the valuation of the Omega software is summarized as follows:

1. The analyst is provided the COCOMO variables that correspond to each program in the Omega software, as presented in Exhibit 4.
2. The analyst matches the provided COCOMO variables for each program to the values in the COCOMO equation, as presented in Exhibit 3.
3. The analyst is provided with the SLIM primary trend group for each program in the Omega software, as presented in Exhibit 6.
4. The analyst is provided with logical executable source lines of code for the software, as presented in Exhibits 5 and 6.
5. The analyst inputs the indicated effort multiplier and scaling exponent, and the provided logical executable lines of source code into the COCOMO II post-architecture equation to estimate the number of person-months to replace each program, as presented in Exhibit 5.
6. The analyst inputs the logical executable source lines of code for each of the programs into the SLIM model to estimate the number of person-months to replace the program, as presented in Exhibit 6.
7. The analyst makes an adjustment for the obsolescence to any programs that are scheduled to be retired, as presented in Exhibits 5 and 6. The functional obsolescence adjustment is based on the expected retirement date and the subject software's UEL.
8. To simplify this illustrative example, let's assume that there is no economic obsolescence related to the Omega total unit of operating property. Therefore, the analyst does not have to apply any economic obsolescence adjustment.
9. The analyst estimates the software person-month development effort based on the average of the RCNLD development effort in person-months indications from the two development effort estimation models: COCOCO II and SLIM, as presented in Exhibit 1.
10. The analyst is provided with the head count and associated costs related to the Omega software development personnel, as presented in Exhibit 2.
11. The analyst applies a 5 percent developer's profit and a 12 percent entrepreneurial incentive to reflect the profit motive and opportunity cost associated with developing the Omega software, as presented in Exhibit 2.
12. The analyst calculates the full absorption cost per person-month, as presented in Exhibit 2.
13. The analyst multiplies the full absorption cost and the average development effort in person-months (estimated using the development effort estimation models) to arrive at the RCN of the Omega software, as presented in Exhibit 1.
14. Other than the functional obsolescence components already considered in the RCN calculation, the analyst concluded that there is no additional functional obsolescence.
15. The analyst concluded that there is no economic obsolescence associated with the ownership and operation of the Omega software.

As presented in Exhibit 1, the analyst concludes that the value of the Omega software, as of the January 1, 2019, valuation date, is $23 million (rounded).

Effect on the Property Tax Assessment

The value of the Omega total unit of operating property—that is, tangible property and intangible property—was estimated as $100 million. This taxpayer total unit value included the value of the software intangible personal property.

As presented in Exhibit 1, the value of the software was $23 million as of the January 1, 2019, valuation date. Subtracting the value of the software intangible personal property yields a value of $77 million ($100 million total unit value less $23 million intangible personal property) in order to conclude the $77 million value of the Omega taxable tangible property as of January 1, 2019. Therefore, the software valuation resulted in reducing the Omega SALT assessment by more than 20 percent. ■