Journal of Multistate Taxation and Incentives (Thomson Reuters/Tax & Accounting)
Volume 28, Number 6, September 2018

PROPERTY TAX

Using the Cost Approach to Value Internally Developed Computer Software for Property Tax Purposes

Taxpayers in jurisdictions that tax only tangible property should ensure the value of internally developed computer software is excluded from the value of assets subject to property taxation.

By CONNOR J. THURMAN

CONNOR J. THURMAN is an associate at Willamette Management Associates in the firm's Portland, Oregon, office and can be reached at (503) 243-7514 or at cjthurman@willamette.com.

In some taxing jurisdictions, the internally developed computer software of a taxpayer company may be exempt from state and local ad valorem property taxation. In these situations, the property tax assessment should not include the value of the taxpayer's internally developed computer software. Let's assume that the taxpayer is the type of company that is subject to property taxation based on the unit principle of property valuation. In that case, the unit value conclusion typically includes the value of all of the taxpayer's tangible property and the value of 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 total unit value for the value of any exempt intangible personal property (such as internally generated computer software).

This discussion focuses on generally accepted methods that valuation analysts may use to value internally developed computer software for property tax purposes. Specifically, this discussion focuses on the application of the cost approach, and the replacement cost new less depreciation method, to value internally developed computer software.

Introduction

Many taxing jurisdictions tax the value of commercial taxpayer intangible personal property for ad valorem taxation purposes. That is, some taxing jurisdictions tax all of the tangible property and all of the intangible property of commercial taxpayer companies. In these taxing jurisdictions, a taxpayer company's internally developed computer software intangible asset would be subject to state and local ad valorem property taxation.

However, some taxing jurisdictions only tax the tangible property—that is, the real estate and/or tangible personal property—of commercial taxpayers. In these jurisdictions, the value of a taxpayer company's intangible personal property (including internally developed computer software) is exempt from ad valorem property taxation. Commercial taxpayers in these jurisdictions—especially commercial taxpayers subject to the unit principle of property valuation—should ensure that the value of their internally developed computer software is excluded from the value of the total bundle of assets subject to property taxation.

This discussion focuses on the valuation of internally developed computer software as intangible personal property. There are generally accepted cost approach, market approach, and income approach methods that may be used to value internally developed computer software source code. This discussion focuses on the application of the cost approach and, in particular, the replacement cost new less depreciation ("RCNLD") intangible personal property valuation method. The RCNLD method is commonly used to value commercial taxpayer internally developed computer software source code and associated documentation and databases.

This discussion (1) describes computer software and (2) presents an overview of the cost approach, RCNLD method. For the valuation of computer software, valuation analysts ("analysts") may use software development effort estimation models to determine the approximate amount of time required to replace the subject software. In particular, this discussion focuses on the COCOMO model and the SLIM model (defined later in this discussion). This discussion also presents an illustrative example of the application of the cost approach, RCNLD method, to value the taxpayer's internally developed software and associated intangible property.

Definition of Computer Software for Property Tax Purposes

Computer 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 computer hardware. In Revenue Procedure 69-21, the Internal Revenue 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 subject computer software is taxable

The determination of whether computer 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 an inconsistent collection of state-specific rules and methods by which analysts and tax advisers contend for guidance in determining what portion (if any) of a taxpayer's computer software assets is taxable and what portion is exempt from property taxation.

When valuing computer software for property tax purposes, it may be important to determine whether the subject software is taxable or tax exempt. Most taxpayer companies own and operate software that has been either:

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

Some states assess property taxes on internally developed computer 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. Therefore, taxpayer companies take the position that the source code and related documentation of the computer software is intangible personal property and should be exempt from property taxation.

Three general lines of reasoning have been devised 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 company purchased a tangible storage medium versus the intangible knowledge contained within
2. whether the subject computer software is operating (or "operational") software or application software
3. whether the subject computer software is internally developed or "bundled"

Line of reasoning one

The first line of reasoning, which we may call 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

Intangible information in this context refers to the digital manifestation of human knowledge in the form of computer code, which 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:

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

The container test may be less relevant in the modern computing environment. This is because the use of a tangible storage medium for software distribution has declined, and software 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 computer software was considered intangible property and, therefore, not subject to ad valorem property taxation.[3] The court ruled that the computer software was intangible because the "essence of the transaction" was not in the tangible medium that was used to transport the computer software to the consumer (for example a disk or CD-ROM) but rather the computer 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, it was not subject to ad valorem property taxation as tangible personal property.

Line of reasoning two

A number of states have emphasized a second line of reasoning that focuses on how separable the computer software is from the computer hardware on which it operates. Some states insist that computer software is essentially inseparable from the tangible hardware on which it operates.

The Ohio Supreme Court, for example, upheld the Ohio Department of Taxation position that all computer 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 computer software may be subject to ad valorem property taxation.

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

- operating computer software, or
- application computer software.

Operating computer software is generally required in order for the computer to function properly. Sometimes operating computer software is described as "embedded" software or "firmware." This label is based on the fact that the computer 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, however, operating software may have a more expansive definition that includes a general-purpose OS that works in conjunction with the BIOS.

The Kansas Department of Revenue describes 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]

Further, the California State Board of Equalization states as follows:

> 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 computer first executes a BIOS when the laptop computer is turned on. In some taxing jurisdictions, this BIOS may be considered tangible personal property that is subject to property taxation. Once the laptop computer has started operating, a user may choose to execute an application such as Microsoft Office.

Microsoft Office may qualify as tax-exempt application software. This is because it executes "on top" of the BIOS and is not required for the computer to operate (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 tax-exempt 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 offers a useful guideline, but it is only a general guideline. Not all operating software is subject to property tax and not all application software is tax exempt. The analyst should perform sufficient due diligence to determine whether the subject software is subject to property tax or is tax exempt.

Line of reasoning three

The third and final line of reasoning classifies computer software as either:

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

Bundled computer software typically includes computer 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 computer software suite. If company ABC 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 taxing jurisdiction exempts bundled computer software), while the value of the laptop computer would be included as tangible property.

This concept is fairly consistent with the operational software/application software dichotomy. The distinction in this line of reasoning becomes more evident if one considers that company ABC may be taxed on its laptop computer software if it instead internally develops an application with word processing and other office productivity features.

Taxability, under the third line of reasoning, 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 taxability of the subject computer software when the analyst considers the many ways in which software can be created, modified, and distributed. If a software developer is tasked to create 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 software for a chain of franchise businesses and then licenses the software individually to 100 franchisees, some taxing jurisdictions may classify the computer software as having been developed for commercialization. This may be true even though the customers belong to the same franchise chain.

Computer Software Valuation Approaches and Methods

There are three generally accepted intangible personal property valuation approaches. These three generally accepted valuation approaches are summarized below.

Cost approach

The cost approach estimates the value of an intangible personal property as the cost (in terms of current dollar expenditures) required to create an intangible asset with equivalent utility and functionality as the subject asset. 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 asset is superior to the subject asset, then allowances may be made for the various forms of obsolescence, including functional (including technological) obsolescence and external (including economic) obsolescence.

Market approach

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

Income approach

The income approach recognizes the prospective revenue, expenses, profitability, and investments associated with the ownership of an intangible personal property. This approach estimates the value of an intangible asset 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 it may be estimated over the asset's expected remaining useful life ("RUL"). 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 will focus on the application of the cost approach, and specifically the RCNLD method.

Cost Approach

The cost approach is based on valuing software based on some measure of cost. The common types of cost that may be estimated within the cost approach include the following:

- the reproduction cost new ("RPCN")
- the replacement cost new ("RCN")

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

The RCN refers to the cost to create the functionality or utility of the subject software, but in a form or appearance that may be quite different from the subject software. While the replacement software performs the same task as the subject software, the replacement software is often superior (in some way) to the subject 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.

Two methods that may be used to estimate the RPCN or RCN of computer software are (1) the trended historical cost method and (2) the software engineering development effort estimation model method.

The trended historical cost method

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 may include all costs associated with the development of the subject software.

An allocation of taxpayer company 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 may include an allowance for the software developer's profit on the software development project, an allowance for entrepreneurial incentive to

motivate the software 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 RPCN of the subject software. In many cases, due to technological advances in programming languages or programming tools, for example, the RCN for the subject software may be lower than the RPCN for the subject software.

Software engineering development effort estimation models

The analyst may employ software engineering development effort estimation models in order to estimate either the RPCN or the RCN of the taxpayer internally developed software. Generally, software engineering development effort measurement models were originally developed to assist software developers in estimating the effort, time, and human resources needed to complete a software project. These models have been adapted by analysts for internally developed software valuation purposes.

The primary input to the software engineering cost estimation models is a size-related metric. Capers Jones, an authority in the field of 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. guessing at the size using "project manager's intuition"
5. guessing at the size using "programmer's intuition"
6. sizing using statistical methods or Monte Carlo simulation[8]

Historically, the most common sizing metric has been the number of software program lines of code. The definition of a line of code and the associated line of code counting conventions vary among the common software engineering development effort estimation models.

A common definition of a line of code is as source code instructions (i.e., instructions as written by human programmers) 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 computer language being 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 valuation of internally developed software can also be developed using different base size units than source lines of code. Examples of these include both function points and object points.

Two common software engineering development effort estimation models are 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 because they generate effort estimates using a set of quantified inputs, such as 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 software engineering development effort estimation models calculate an estimate of the effort required to develop a software system in terms of person-months. The number of person-months is multiplied by a blended cost per person-month to arrive at the indicated value of the software. The blended cost per person-month is typically a full absorption cost (e.g., the cost of a software programmer would include benefits, wages, applicable overhead, etc.).

Additional software engineering development effort estimation models include (1) the KnowledgePlan ("KPLAN") model and (2) the SEER for Software ("SEER-SEM") model.

KPLAN

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

The specific algorithms utilized by KPLAN have not been fully disclosed. 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 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 software engineering development effort estimation model. The model is still available for download from various software archive websites.

SEER-SEM

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

The following is a list of the specific SEER-SEM models and the questions they address:

- sizing (how large is the project?)
- technology (how productive are the developers?)
- effort and schedule calculation (what amount of effort and time is needed?)
- constrained effort/schedule calculation (how does the expected outcome change with constraints?)
- activity and labor allocation (how should tasks and labor be allocated?)
- cost calculation (given effort, duration, and labor, how much will the project cost?)
- defect calculation (what is the expected quality of the delivered computer software?)
- maintenance effort calculation (how much maintenance will be required?)
- progress (how is the project progressing and is it on track to target completion?)
- validity (is the project feasible based on the technology involved?)

The current version of SEER-SEM (version 7.3) is the first version of the model to incorporate all stages of the project estimate's life cycle. The model relies on parametric modeling that also utilizes 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 by Barry W. Boehm, PhD, and is described in *Software Engineering Economics.*[9] This development effort estimation model projects the amount of effort required to develop the 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 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:

- adjusting the coefficients in the equations, and
- 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. I will provide an illustrative example of a cost approach valuation analysis using COCOMO II later in this discussion.

The post-architecture software development equation defined by the COCOMO II model is 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 calculated 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 software engineering development effort model was developed by Lawrence Putnam, the founder of 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 academic literature as the "Putnam model") estimates the amount of effort in person-months required to develop software based on the following parameters:

- a project size build-up parameter (a number representing a range from entirely new software to rebuilt software)
- the software delivery time
- the effort required to create the computer software

- the expected rate of defective software
- a productivity environment factor

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

The SLIM model allows users to specify the given computer software project's environment by identifying the industry function for which that computer software will be used. The SLIM model utilizes a primary trend group to benchmark the subject 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.

This discussion presents an illustrative development effort estimation analysis output using the SLIM model below.

Source lines of code adjustments

As discussed previously, the software engineering development effort estimation model method often relies on an input of source lines of code to determine the amount of effort needed to replace the internally developed software. The analyst may need to make adjustments to company-provided source lines of code. These 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 to write large quantities of code, software 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 internally developed 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 source lines of code

The specific line of code size measure used by both COCOMO II and SLIM is logical executable lines of code. In order to define logical executable lines of code, the following paragraphs explain:

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

A physical line of code may be thought of as:

- one line as typed by a programmer (i.e., before deliberately beginning a new line), or
- 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 programmer to spread one logical program instruction over two or more physical lines. Some programming languages allow the programmer 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 valuing internally developed software for property tax purposes, the analyst should make any necessary adjustments for all forms of obsolescence. Adjustments are made to the various cost estimate in order to account for losses in value resulting from:

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

These three types of property obsolescence are summarized below:

Physical deterioration is a loss in value of the taxpayer operating assets 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 of the taxpayer operating assets caused by the inability of the subject property to adequately perform the function for which it is utilized. Functional obsolescence is, therefore, internal to the subject property. Functional obsolescence is often related to such factors as property super-adequacies, excess property operating costs, and property inadequacies.

External obsolescence is a loss in value of the taxpayer operating assets 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 subject property.

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

When a reproduction cost new method, such as the trended historical cost method, is used to value software, technological obsolescence can be significant. This factor is due to increasing productivity and technological advances over time. The use of a replacement cost new method typically eliminates the productivity-related technological obsolescence. However, other adjustments for technological obsolescence may be necessary. Economic obsolescence usually has more relevance with respect to product software. However, this form of obsolescence may be examined in the valuation of operational software as well.

Although the value of tangible personal property is often estimated using depreciation schedules, properly maintained computer software 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, any attempt to estimate obsolescence for properly maintained software by "depreciating" it over some finite time period may be unsupportable.

Remaining useful life analysis

The estimation of the RUL may be an important consideration in each of the three generally accepted approaches to software valuation. In the cost approach, an RUL analysis may be performed in order to estimate the total amount of obsolescence, if any, from the estimated measure of cost—that is, either reproduction cost, replacement cost, or trended historical cost.

The analyst's assessment of RUL may have a measurable effect on the value of the software. Normally, a longer RUL would indicate a higher value for the subject taxpayer software and a shorter RUL would indicate a lower value for the subject taxpayer software.

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 development 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 necessary adjustments for (1) developer's profit and (2) entrepreneurial incentive into the full absorption cost estimate. A discussion of these adjustments follows.

Developer's profit

Developer's profit is the expected return an intangible asset developer expects to receive over the direct 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 and indirect costs to replace the internally developed software systems.

The analyst may utilize selected guideline publicly traded companies in the computer programming services industry to identify a reasonable developer's profit. One method of analysis is to compare 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 using the following formula:

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

The developer's profit margin that is the result of this formula is a percentage that is applied to the direct and indirect cost of development to calculate the total direct cost, indirect cost, and developer's profit. An example of this calculation follows.

Operating profit that is 7.7% greater than the total cost of development is mathematically equivalent to a profit margin of 7.1% (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%.

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 an entrepreneurial incentive cost component by considering the following:

- a rate of return, as indicated by the taxpayer management
- the estimate of the amount of time required to replace the subject internally developed software, as indicated by the subject taxpayer management
- the sum of the estimated software developer's profit and direct and indirect replacement costs incurred during the estimated time required to replace the internally developed computer software

The entrepreneurial incentive considers management estimates of the time required to replace the subject internally developed software.

Illustrative Software Valuation Example

Let's assume that Alpha Gas Transmission Company ("Alpha") is an intrastate natural gas pipeline company. Alpha is assessed in its taxing jurisdiction based on the unit principle of property valuation. Let's further assume that the assessor values the Alpha total unit of operating property at $100 million as of January 1, 2018. Let's also assume that intangible personal property is exempt from property taxation in the subject taxing jurisdiction. Alpha owns internally developed computer software that is used to operate its compressor stations and its pipeline operations.

Alpha retained an analyst to estimate the value of this internally developed software so that the taxpayer can remove the value of that intangible personal property from the total unit value. The analyst decided to use the cost approach and the RCNLD method to estimate the value of the Alpha subject software as of January 1, 2018. To simplify this illustrative example, let's assume that computer software is the only intangible personal property that is owned and operated by Alpha as of January 1, 2018.

Summary of accompanying 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 Alpha computer software. This analysis includes associated direct and indirect costs, as well as the selected developer's profit and entrepreneurial incentive applicable to the Alpha software development personnel.

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 taxpayer software programs.

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 taxpayer software.

Cost approach—replacement cost new less depreciation method

The simplified process of how the analyst performs the valuation of the Alpha software is as follows:

- The analyst is provided the COCOMO variables that correspond to each software program in the subject Alpha software, as presented in Exhibit 4.
- The analyst matches the provided COCOMO variables for each software program to the values in the COCOMO equation, as presented in Exhibit 3.
- The analyst is provided with the SLIM primary trend group for each software program in the subject Alpha software, as presented in Exhibit 6.
- The analyst is provided with logical executable source lines of code for the subject software, as presented in Exhibits 5 and 6.
- 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 determine the person-months to replace each software program, as presented in Exhibit 5.

- The analyst inputs the logical executable source lines of code for each of the software programs into the SLIM model to determine the person-months to replace the program, as presented in Exhibit 6.
- The analyst makes an adjustment for the obsolescence to any software 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 RUL of the software program.
- To simplify this illustrative example, let's assume that there is no economic obsolescence related to the Alpha total unit of operating property. Therefore, the analyst does not have to apply any economic obsolescence adjustment to the cost approach valuation of the software intangible personal property.
- The analyst estimates the subject computer software person-month development effort based on the average of the RCNLD development effort in person-months indications from the two software engineering development effort estimation models: COCOCO II and SLIM, as presented in Exhibit 1.
- The analyst is provided with the head count and associated costs related to the Alpha software development personnel, as presented in Exhibit 2.
- The analyst applies a 5% developer's profit and a 12% entrepreneurial incentive to reflect the profit motive and opportunity cost associated with developing the subject Alpha software, as presented in Exhibit 2.
- The analyst calculates the full absorption cost per person-month, as presented in Exhibit 2.
- The analyst multiplies the full absorption cost and the average development effort in person-months (estimated using the software engineering development effort estimation models) to arrive at the RCNLD of the subject Alpha software, as presented in Exhibit 1.

As presented in Exhibit 1, the analyst concludes that, based on the estimated effort, the value of the Alpha internally developed software, as of the valuation date, is $23 million (rounded).

Effect on the Property Tax Assessment

The value of the Alpha total unit of operating property—that is, tangible property and intangible property—was estimated as $100 million. However, this total unit value included the value of the subject software intangible personal property. As presented in Exhibit 1, the value of the subject software was $23 million as of the valuation date. Subtracting the value of the subject 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 Alpha taxable tangible property as of January 1, 2018. Therefore, the software valuation analysis resulted in properly reducing the Alpha property tax assessment by more than 20%.[13]

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

[2] Rulings of the Tax Commissioner, Virginia Dep't 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*, 2d ed. (New York: McGraw-Hill, 2007) at 8.

[8] *Id.* at 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) at 229.

[13] In addition to the sources for this article cited in the preceding footnotes, the author would like to acknowledge the following: John E. Elmore, "The Valuation of Computer Software in the Health Care Industry," Willamette Management Associates *Insights* (Summer 2016).

Exhibit 1. RCNLD Method Valuation Summary

Alpha Gas Transmission Company Internally Developed Computer Software Programs
Cost Approach – Replacement Cost New Less Depreciation Method Valuation Summary
As of January 1, 2018

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	1,128	Months
Selected Subject Software Person-Month Development Effort Estimate [b]		1,807	Months
Subject Software Person-Month Development Effort Estimate		1,807	
Full Absorption Cost per Person-Month	2	$12,700	
Subject Software Replacement Cost New less Depreciation Indication		$22,954,945	
Subject 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.

Sources: As indicated above.

Exhibit 2. Full Absorption Cost Per Person-Month

Alpha Gas Transmission Company
Internally Developed Computer Software Programs
Cost Approach – Replacement Cost New Less Depreciation Method
Software Development Personnel
Full Absorption Cost per Person-Month
As of January 1, 2018

Software Development Actual Cost Components	Software Development Personnel
Actual Headcount:	132
Actual Costs:	
Salaries	10,500,000
Employee Benefits	2,625,000
Bonuses	525,000
Overhead	3,412,500
Total Actual Annual Cost	17,062,500
Monthly Cost per Person:	
Total Actual Annual Cost	17,062,500
Divided by: Headcount	132
Annual Cost per Person	129,261
Divided by: 12 Months	12
Direct and Indirect Cost per Person-Month	10,772
Computer Software Developer's Profit [a]	5%
Direct Cost, Indirect Cost, and Developer's Profit per Person-Month	11,310
Direct Cost, Indirect Cost, and Developer's Profit Cost per Person-Month	11,310
Entrepreneurial Incentive as a Percent of Direct Cost, Indirect Cost, and Developer's Profit [a]	12%
Full Absorption Cost per Person-Month	12,668
Full Absorption Cost per Person-Month (rounded)	12,700

[a] Determined by the analyst (details not presented).

Source: Taxpayer-provided costs and headcount and analyst calculations.

Exhibit 3. COCOMO II.2000 Variables

Alpha Gas Transmission Company Internally Developed Computer Software Programs
Cost Approach – Replacement Cost New Less Depreciation Method
COCOMO II.2000 Variables
As of January 1, 2018

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
CMM	*1(LH)*	*1(UH)*	*2*	*3*	*4*	*5*
Constants						
Multiplicative Constant (A)	2.94					
Exponential Constant (B)	0.91					
As of Date	1/1/2018					

Exhibit 4. COCOMO II.2000 Effort Multipliers and Scaling Exponents

Alpha Gas Transmission Company
Internally Developed Computer Software Programs
Cost Approach – Replacement Cost New Less Depreciation Method
COCOMO II.2000 Effort Multipliers and Scaling Exponents
As of January 1, 2018

		Computer Software Programs								
		Program 1			Program 2			Program 3		
	Software Development Cost Drivers	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		1.10
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	1.00		L	0.87		VL	0.73	
	Complexity - Computational Operations	VL	0.73		L	0.87		N	1.00	
	Complexity - Device-Dependent Operations	N	1.00		N	1.00		L	0.87	
	Complexity - Data Management Operations	N	1.00		N	1.00		N	1.00	
	Complexity - User Interface	VL	0.73		L	0.87		N	1.00	
RUSE	Required Reusability	N		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 Alpha software development personnel.

Exhibit 5. Development Effort—COCOMO II Model

Alpha Gas Transmission Company
Internally Developed Computer Software Programs
Cost Approach – Replacement Cost New Less Depreciation Method
Development Effort – COCOMO II Model
As of January 1, 2018

Software Application	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	625,000	0.25	1.0631	690	0%	-	690
Program 2	485,000	0.41	1.0749	929	20%	186	743
Program 3	355,000	0.56	1.1005	1,055	0%	-	1,055
	1,465,000			2,673		186	2,487

[a] Alpha management provided the logical executable source lines of code for the subject software.
[b] As presented in Exhibit 3.
[c] A 20 percent obsolescence adjustment was applied for program 2 based on eight years remaining of a 10 year RUL of the program, as indicated by Alpha IT personnel.

Exhibit 6. Development Effort—SLIM Estimate Model

Alpha Gas Transmission Company
Internally Developed Computer Software Programs
Cost Approach – Replacement Cost New Less Depreciation Method
Development Effort – SLIM Estimate Model
As of January 1, 2018

Software Application	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	482	0%	-	482
Program 2	Business	485,000	402	20%	80.4	322
Program 3	Business	355,000	324	0%	-	324
		1,465,000	1,208		80	1,128

[a] Based on the planned use and function of the subject software programs.
[b] Alpha management provided the logical executable source lines of code for the subject software.
[c] Derived by the analyst using 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 RUL of the program, as indicated by Alpha IT personnel.