Software & Valuation In The Information Society

Part Three: The Software Inventory Valuation—TSV (OV)

By Dwight Olson

Software based intellectual assets are known in the computer industry as the software inventory. This software inventory could be considered to include both Intellectual Property (IP) and Intellectual Assets (IA). Like Vertex of Canada one could consider traditional IP to comprise just patents, trademarks, copyrights and industrial design registrations because these four Intellectual Properties have legislation to protect the legal owners in the disclosure of these assets.[1] Non disclosed Intellectual Assets (IA) in the software inventory then could be considered to include trade secrets, unpublished copyrights, and know-how or the codified, tangible descriptions of specific knowledge which a business uses to support commercialization. Although Canada does not have specific laws that apply specifically to trade secrets, U.S. state jurisdictions do.[2] Trade secrets, unpublished copyrights, and know-how are those software inventory assets that contain valuable proprietary information that belongs exclusively to the business, and which is used in the business to provide economic or competitive advantage. Many proprietary software vendors consider trade secrets to include unpatented inventions, formulas, processes, devices, patterns, designs, design drawings, source code, customer databases, and internal operations manuals. However, in the open source world, disclosure of the source code is covered by copyright protection.

Many investments today are secured using traditional IP such as software patents belonging to the organization, but seldom are unpublished copyright, know-how and trade secrets used.[3] This is most interesting because for many industries, including the software industry, the trade secrets and non disclosed software inventory are in many cases more valuable than the traditional IP.

According to Karl Jorda, there are deep–seated misconceptions about the relationship between traditional IP, especially patents and non-traditional IA or trade secrets.[4] These misconceptions are very prevalent. Trade secrets are treated as the orphan or the black sheep in the IP barnyard.[5] They are maligned as flying in the face of the patent system, the essence of which is disclosure of inventions to the public. Keeping inventions secret is, therefore, supposed to be reprehensible. One noted IP professor in Washington went even so far as to say: "Trade secrets are the cesspool of the patent system."[6] Karl Jorda suggests nothing could be further from the truth. Trade secrets are the "crown jewels" of corporations; they are the IP of the new millennium and can no longer be treated as a stepchild."[7] Indeed, trade secrets are now gaining greater reverence as a tool for protection of innovation. The stakes are getting higher with injunctions becoming a greater threat in trade secret misappropriation cases and related damage awards in the hundreds of millions. In a trial in Orlando, Florida, two businessmen sought $1.4 billion in damages from Walt Disney Co. after accusing the company of stealing trade secrets for the sports complex at Walt Disney World. In this case, the jury awarded them $240 million. And the misappropriation of genetic corn seed material trade secrets owned by Pioneer Hi-Bred International by Cargill, Inc. cost the latter $300 million.[8]

■ Dwight Olson, CLP,
LESI Copyright Committee,
V3Data, Principal,
San Diego, CA, USA
E-mail: dcolson@ix.netcom.com

As trade secrets, know-how and other intangibles become more recognized as assets including software; one might ask the question, "How do

1. Intellectual Asset Identification, The First Step in an Intellectual Property Management Program, Dave Tyrrell and Gary Floyd, Vertex Intellectual Property Strategies Inc.

2. Ibid.

3. Leveraging Software Via The Capital Markets, *les Nouvelles*, September 2008, Dwight Olson and David Drews.

4. Intellectual Property Valuation, The Legal Counterpart/Counterpoint, Karl F. Jorda David Rines Professor of Intellectual Property Law & Industrial Innovation, 2004.

5. Ibid.

6. Ibid.

7. Ibid, Mark Halligan.

8. Ibid.

you value them when they are to be considered as a financial asset?" Data Securities International introduced a software valuation concept Total Software Value (TSV) in the mid 1980's. This valuation concept uses the Ownership Value (TSV-OV) (the IA software inventory), Market Value (TSV-MV), and Internal Cost Savings (TSV-ICS) as values and influencing variables when considering software as a financial asset.[9] A software inventory valuation or TSV-OV analysis looks at the sum total of all the various IA software components (see Table 1) that make software licensable as a software product. The interim TSV-OV sum total value is then biased by influencing factors such as the FASB risk factors,[10] to more closely project a fair value for the product's software inventory. A software market valuation or TSV-MV analysis uses a discounted cash flow method to compute an interim TSV-MV value. Then to this interim value the same influencing factors from the software inventory analysis are applied to determine a software fair market value. A software valuation for internal cost savings or TSV-ICS analysis again looks at the sum total (or bundle) of the various software components that should be present to make software usable within an enterprise to determine an anticipated license value and subsequent cost savings. This article will take a closer look at the TSV-OV valuation process and subsequent articles will discuss TSV-MV and TSV-ICS valuations.

TSV Software Inventory Valuation (OV)

Much of software is made up of know-how, trade secrets, and unpublished copyrights that are often used to describe the same software component.[11] It is during development and commercialization of the software inventory for a software product or a software product line where software asset components come into existence and these software patents, trademarks, copyrights, know-how, and trade secret components emerge and sometimes collide.[12] There is often confusion as to what software intellectual assets to include in the valuation and how to value each type of software asset. For example, at the first launch of a software product, can one really believe the forecasted number of licenses and then compute a market value based on forecasted future sales? Does a newly registered trademark really have any value for a non-established software product? And, for software patents, is there really any blocking monopoly value yet? Why would one want a monopoly if a market is not yet established? It seems as though one would want as many competitors in the field as possible helping to establish the market. What is important to remember in commercialization of a software product is that only after the market has been established and demand is apparent, the software inventory contains the only assets with value. So even if one is valuing an open source product, the value is also in the software inventory just as in a proprietary product.

Aura Soininen said in his thesis, "The value of a patent is dependent on the value of the invention it claims, and how it is commercialized. Thus, most patents are worth very little, and most inventions are not worth patenting: it may be possible to protect them in other ways, and the costs of (patent) protection may exceed the benefits."[13] Thankfully, software patent protection does not require the disclosure of all background technology and related components necessary for market acceptance and penetration. For proprietary software the disclosure of these components may quickly decrease the proprietary software asset values and may also increase immediate competition. However, in the open source community, disclosure of the source code may increase market penetration and acceptance. In either case, any trade secrets, know how, and other background components not disclosed are considered proprietary and for internal use only by their owner. For a TSV OV the software inventory components need to be identified, just as all other "hard" assets and valued. Table 1 lists the various components that are typically produced during the development cycles for a software product.[14] One of the software asset components called the "source code" is typically the most costly to produce, test, and document, and thus arguably the most valuable. We will use this particular component to further illustrate TSV OV valuation.

Gordon Smith in his IP valuation book suggested that one method to measure fair market value is its

9. Software & Valuation in the Information Society, *les Nouvelles*, June 2008.

10. Ibid.

11. Software & Valuation in the Information Society Part 2, *les Nouvelles*, December 2008.

12. Ibid.

13. Patents in the Information and Communications Technology Sector—Development Trends, Problem Areas and Pressures for Change, Aura Soininen, ISBN 978-952-214-344-0 (PDF) 2007.

14. Software & Valuation in the Information Society, Part 2 the Software Inventory, *les Nouvelles*, December 2008, Dwight Olson and Denny Kolb.

Table 1. Software – Intangible Asset Inventory

Marketing and Sales
- Marketing plans and collateral

Client Support Systems
- Installation and training
- User documentation and help
- Client databases

QA and testing
- Bug/support system
- Testing code and data

Manufacturing System
- Specific build guides

Commercialization Strategy
- Product plan (release and updates)

License Management
- DRM and license controls
- Back office system

R&D Systems
- Internal design documentation
- Source code with comments
- Source code control with comments
- ASP databases
- QA test and procedures
- Third party software
- Open source and strategy
- Client databases

cost to replace.[15] Typical financial valuations have historically examined the financial statements for either the capitalized software costs,[16] defined by Financial Accounting Standards Board No. 86, or attempted to break out a portion of R&D expenses that could be allocated to the software. A much more rational approach is to ask the experts that develop software to value the software inventory. In other words, what do they predict the cost to develop a component would be or would have to be paid to have a similar software asset built? This value can be calculated using software costing models that bias the projected cost based on actual historical data. For example, a TSV OV valuation for source code with comments is best done using an estimated replacement cost method for determining a reasonable fair market value. But unlike most valuations that only value to the first release, a TSV valuation can be designed to look at all of the developments and processes that burn through the vast majority of money involved in the development of software. Indeed, by using this technique a meaningful fair market valuation can be done at any version release point in the life cycle of a software product.

A TSV valuation for the software inventory is to look at the results of a software project estimation model to determine if its projected estimate is reasonable for a fair market value. According to the experts in software cost modeling, experience to date suggests that no single technique is best for all development contexts. If we were only using a software cost model to project cost, then a comparison of results of several approaches is most likely to produce a more realistic cost estimate. This is suggested by the experts.

Typically software engineering cost models and estimation techniques are used for a number of purposes. These include budgeting, tradeoff and risk analysis, project planning and control, cost and schedule breakdowns by component, and software improvement investment analysis. TSV OV expands the list to include the use to value the software inventory such as the source code. Again, in selecting the cost model it is recommended using one that biases the cost based on historical data. The purpose of such is to determine if the economic value computed by the cost model is reasonable compared to similar efforts. There are a number of databases that contain information on similar efforts such as COCOMO[17] (COnstructive COst MOdel), as well as other sizing data such as that associated with the verification results which has been gathered by Data Securities International[18] over its 25 year history.

Increasing Value of Software

Before we can proceed to further discuss the inventory valuation of software, one must consider what happens to software value over time during its life cycle in the marketplace. It is here where software differs crucially from other intangible and tangible assets. Successful software products typically have many versions, long lifetimes, and corresponding high maintenance cost ratios over their lifetime. Software lifetimes before complete product (not version) substitution are 10 to 15 years, and are likely to increase.[19] New version release frequency is usually determined by the rate of required bug fixes needed and the tolerance of users to dealing with upgrades. Many examples have shown an average rate of 18 months for new versions. With well-maintained software, in active use, it does not wear out, and is likely to gain value.[20] The majority of software costs

15. *Valuation of Intellectual Property and Intangible Assets*, Third Edition, Gordon Smith and Russell Parr.

16. Software Valuation in the Information Society, *les Nouvelles*, June 2008.

17. *Software Development Cost Estimation Approaches–A Survey*, Barry Boehm, Chris Abts University, Sunita Chulani IBM Research.

18. Now Iron Mountain.

19. *What is Your Software Worth?* Gio Wiederhold, Stanford University, April 2007.

are incurred during the period after the first release to the marketplace and accepted by industry.

Accordingly, successful enterprise software products have many versions, long lifetimes, and corresponding high maintenance cost ratios over their lifetime. Maintenance costs of such enterprise software amount to 60 percent to 90 percent of total costs.[21] These costs are primarily due to software maintenance, which refers both to the activities to preserve the software's existing functionality and performance, and activities to increase its functionality and improve its performance throughout the life-cycle.[22] Thus, over that life, there may be 10 significant version releases. Early in its life, there may have been several versions per year. Software that is significantly dependent on external conditions will require more frequent updates.

In "What is Your Software Worth" by Gio Wiederhold, it is suggested that one should measure code sizes of software versions to allocate its relative contribution to the IP. This approach assumes that the value of a unit of the original code is just as valuable as a unit of new code. Wiederhold suggests there are valid arguments that code size is not a surrogate for the contents of the IP. One argument is that later code, being more recent, represents more recent innovation, and hence should be valued higher. An argument in the opposite direction is that the basic functionality is represented by the initial code. There may have been a few lines of brilliant initial code, slowly buried in a mass of subsequent system interfaces and later tweaks and fixes, which are critical to the IP. The architectural component of software also represents valuable IP or IA and changes little over the life of the software. Wiederhold suggests one might find that much code is inserted later to deal with error conditions that were not foreseen originally. Code that was added during maintenance has its value mainly in terms of providing smooth operation and a high degree of reliability. Usually the original code provided the functionality that motivated the customer's acquisition in the first place. If that functionality had been inadequate, the customer probably will move to a subsequent version. However, newer versions of the code will likely include adaptations and perfections that will motivate additional sales. Thus, given that the positives and negatives can balance each other out, Wiederhold found that it is reasonable to assign the same value to lines of old and new code. In TSV valuation, we have found it reasonable to use relative code size as a surrogate to measure value in a piece of software. As long as the methods to obtain metrics are used consistently and without bias, the numbers obtained will be adequate for the inherently difficult objective of valuing software.

Chris Kemerer suggests that a distinct disadvantage of any formal model is the inconsistency of estimates. He conducted a study indicating that estimates varied from as much as 85-610 percent between predicated and actual values to complete the project.[23] For costing analysis he suggested calibration of the chosen model can improve these figures, however, formal models still produce errors of 50-100 percent.[24] He goes on to say that "one of the most important objectives of the software engineering community has been the development of useful models that constructively explain the development life-cycle and accurately predict the cost of developing a software product. To that end, many software estimation models have evolved in the last two decades based on the pioneering efforts of the above mentioned researchers." Such developments have added greatly to the potential use of such models to predict more accurately fair market values. Especially in those cases when the software product has not achieved market penetration.

TSV Software Source Code Valuation with COCOMO

Significant research on software cost modeling began with the extensive 1965 SDC study of the 104 attributes of 169 software projects that led to some useful partial models in the late 1960s and early 1970s. The late 1970s produced a flowering of more robust models such as SLIM, Checkpoint, PRICE-S, SEER, and COCOMO.[25] This dynamic field of software estimation sustained the interests of those researchers who succeeded in setting the stepping-stones of software engineering cost models and discussed the pros and cons of one software cost

20. Ibid.

21. Thomas M. Pigoski: Practical Software Maintenance - Best Practices for Managing Your Software Investment; *IEEE Computer Society Press*, 1997.

22. *Software Development Cost Estimation Approaches–A Survey*, Barry Boehm, Chris Abts University, Sunita Chulani IBM Research.

23. Reliability of Function Points Measurement: A Field Experiment; *Comm. ACM*, Vol. 36, No. 2, February 1993, pp. 85-97.

24. Ibid.

25. *Software Development Cost Estimation Approaches–A Survey*, Barry Boehm, Chris Abts University, Sunita Chulani IBM Research.

estimation technique versus another, which essentially provided the present analytical techniques.[26]

Two metrics are commonly used in estimating cost projections. They are: Lines Of Source Code and Number of Function Points. In using these metrics to help compute value, the various aspects of programs can easily be measured once the software is at a version release milestone. One can look at Lines Of Source Code or the Number of Function Points. When counting Lines Of Source Code, it is important to not count comment lines, which should be ignored, and multiple statements on a line should be counted only once. For Function Points, software modules that appear multiple times should likewise be counted only once. If modules are written in multiple languages, conversions must be made. There is function-point literature available that provides tables that relate function points to code for many languages.[27] Common procedural languages have ratios of about 100 Lines Of Source Code per Function Point, while Visual Basic is rated at 35 Lines Of Source Code per Function Point. However, Lines Of Source Code may be the most useful for valuation purposes as it is the most common metric used for costing purposes.[28]

As mentioned, an input requirement for an algorithmic model is a metric measure of the size of the finished system. In COCOMO, lines of source code are used, which is obviously not known at the start of a software project. However, in software inventory valuations specifically for the source code component, the number of lines of source code or source lines of code (SLOC) can be easily determined. Likewise, consideration should be given to the fact that SLOC is dependent on the programming language and environment, and should be analyzed within the relevant constraints in the valuation, as noted above. It should be noted that KLOC, or 1000 lines of (source) code, is typically used in reference articles instead of SLOC for easier reference. Using COCOMO to calculate a value for the source code component is a simple linear COCOMO equation to determine the number of man months. Essentially, man months = C*(KLOC)k where C = a constant and k = another constant. The C and k constants are determined in COCOMO to be based on productivity parameters and past efforts.

The COCOMO cost and schedule estimation model was originally published in 1981. It became one of most popular parametric cost estimation models of the 1980s.[29] The COCOMO II research effort was started in 1994 at USC to address the issues on non-sequential and rapid development process models, reengineering, reuse driven approaches, object oriented approaches. COCOMO II was initially published in the Annals of Software Engineering in 1995.[30] The model has three sub models, Applications Composition, Early Design and Post-Architecture, which can be combined in various ways to deal with the current and likely future software practices marketplace. The current model has been calibrated to a database of 161 projects collected from Commercial, Aerospace, Government and non-profit organizations. A primary attraction of the COCOMO models is that the fully-available internal equations and parameter values are available. Over a dozen commercial COCOMO '81 implementations are available; one (Costar) also supports COCOMO II: for details, see the COCOMO II Web site *http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html.*"[31]

Using other information to calibrate the software inventory's SLOC value parameter could be quite helpful to determine the reasonableness of this SLOC value. For example, beginning in 1986, Data Securities International began to value software for debt financing under its verification service and, as part of its verification process, was able to gather information relative to code size of software product deposits in escrow. It is estimated that, to date, the number of verifications exceed 1,000 escrow software product deposit verifications of proprietary software. The data base of SLOC to proprietary software product is proprietary to Data Securities International which is now Iron Mountain, and its corresponding verification company. For other non proprietary software, one could also look at open source projects that pertain to assets of similar function to determine reasonableness of relative size. To date there are over 180,000 open source projects

26. Ibid

27. Jones, C. *Applied Software Measurement*, 1997, McGraw Hill.

28. Ibid.

29. *Software Development Cost Estimation Approaches–A Survey*, Barry Boehm and Chris Abts from the University of Southern California and Sunita Chulani from the IBM Research Lab in San Jose.

30. Ibid.

31. Ibid.

available in Sourceforge[32] alone.

Software costing models generally require estimates of effort and duration. In using the costing model to estimate the software inventory value, the main input is usually the software size. Effort prediction models take the general form of effort = p*S; where p is a productivity constant such as one SLOC per man month and S is the size of the system. For this example, once the value for p is known, one could calculate the cost of the effort.

Let's use COCOMO II to compute an example of the markets anticipated cost for a hypothetical Ultra Lite Unix Data Base Management System (ULUDBMS) software product that had 8500 lines of documented source code at first version release. In this example, 28.9 man months was determined to be the effort by the COCOMO II engine. If we use $20,000 per month for the expected 28.9 person man month effort, then the source code cost to replace would be $578,000. If the associated FASB risk factors were determined to be non-influencing and the SLOC size of 8,500 was also determined to be reasonable using other database information then the TSV OV would be $578,000. This value could also provide some guidance in determining the reasonableness of the actual expenses from a financial statement that might need to be attributed to the other software inventory component.

Given $578,000 was the TSV COCOMO II calculated fair value for this inventory component, one could determine the reasonableness of this estimate using both the actual financials provided as well as other costing methods such as the SLIM model described above. Remember that the source code is just one of the items in the software inventory group. Each item in that group should have a corresponding identification and value component. A TSV OV value for an open source project, would need to be adjusted by any missing internal components (see Table 1) and appropriate FASB risk factors.

TSV OV determined for software developed for commercial use can be valued for a variety of purposes and, under FASB 86, treated as an asset. The risk and quality influencing factors for a TSV OV valuation are technology feasibility, management commitment, financial feasibility, ownership statements, and intellectual property rights.[33] If these factors are computed to be less than one, they would weigh negatively on any TSV OV value proposition.

FASB 86 does not specifically require intellectual property rights, but it became obvious since FASB 86 was implemented over 20 years ago that if you don't own the software, i.e. have the work for hire agreements, the trade secret employee confidentiality, and transfer or assignment of the copyrights and patents for the software product components, you really can't license and make money. The courts have made this very clear. To highlight this, if there is any question as to the ownership of the trade secrets, copyrights, etc. or to the assignment agreements of developers to the enterprise, the resultant TSV OV value diminishes.

In completing a software inventory's TSV OV all

Table 2. COCOMO II Inputs

	Equivalent Size (SLOC)
New	8500
Reused	0
Modified	0
Total	8500

Table 3. COCOMO II Ouputs

Top-level estimate for elaboration & construction		
Effort	28.9 person-months	
Schedule	10.7 months	
Activities (effort in person-months)		
	Effort (person-months)	Schedule (months)
Inception	1.7	1.3
Elaboration	6.9	4.0
Constructions	21.7	6.7
Transition	3.5	1.3

32. *SourceForge.net* is a source code repository and acts as a centralized location for software developers to control and manage open source software development. *SourceForge.net* is operated by Sourceforge, Inc. (formerly VA Software) and runs a version of the SourceForge software, forked from the last open-source version available. As of August 2008, *SourceForge.net* hosts more than 180,000 projects and more than 1.9 million registered users,[2] although it does contain many dormant or single-user projects.

33. Software Valuation in the Information Society, Robert Bramson, *les Nouvelles*, June 2008.

components should be calculated and analyzed to yield a total software inventory value. We can now use the individual values for each of the software inventory components for financial governance or other purposes. Let's assume we had identified and valued all the IA components of ULUDBMS's software inventory and computed the TSV OV fair market value of ULUDBMS to be $1,400,000. We now can also use this TSV OV value in other situations such as determining a commercialized software patent value. Let's assume for discussion, we wish to value a hypothetical software patent that was associated with ULUDBMS.

How Does OV Affect a Software Patent Value?

To show this, let's use the rule of thumb patent valuation that was explained in a 1999 LES article by Bob Bramson.[34] A rule of thumb for a patent value is V = p.v.(X*Y*Z)-$. We use this equation to illustrate the TSV OV benefit of the IA software inventory in a patent valuation in a commercialization (carrot) licensing scenario. In Bramson's rule of thumb; V is the patent value, p.v. means present value, X is the percentage likelihood of commercial success of the software product, Y is the applicable royalty or license rate, Z is the applicable royalty/license base over a period of years, and $ is the cost of completion of development. In Bramson's rule of thumb, for carrot licensing, that is bringing a software product (or software embedded product) to market, the owner must decide whether a patent royalty, a software license, or a combination will be used to define the Y parameter. Assessing the value of X for a software product is easier in some ways and harder in others and a lot depends on the development stage of the software product.[35] It is well known that, if a technology owner is willing to spend the incremental cost of advancing the technology one or two levels closer to commercial reality, the value that the owner receives will increase dramatically and probably more than justify the increased investment.[36] Thus, one could assume that at a minimum the TSV OV would decrease the cost to development. The new equation is simply V=p.v.(X*Y*Z)-$+TSV(OV).

A Valuation Issue of Trade Secrets and Other Know-How Software Components

For proprietary software products, potential sales volume is a key factor required to set a license price for the deliverable components of the software inventory that will provide an adequate future income. The market value of the software to the creator depends on that income potential from licensing, unless the software inventory could be monetized in some other form.[37] Predicting the quantity of licenses associated with this software asset is hard, and expert help is often required. In contrast, an author of a book can obtain this kind of information from a publisher. Once the book is on the market, the interested author is able to track what the total income is from the publisher's annual statements.[38] However, the value to the purchaser of rights to a book or an end user licensee of the software is essentially unaware or at least not concerned with the cost and effort spent to create it. However, if investors were aware of the entire software inventory and their potential values they might be better prepared to make decisions on financial governance.

To conclude, it is also important to understand that one of the most significant issues in dealing with valuation of software components, especially for monetization purposes, is that much of software is trade secrets, unpublished copyrights, and know how. For those components that are patents, trademarks, copyrights and industrial design, federal registries exist and incorporate legislation to protect the investors in these types of assets. Valuation is much easier for traditional IP components when they are identified and registered. It may be time for software based IA or trade secrets and other codified know how to have a privatized and secure registry for the holding of the software components so that business can assert ownership, maximize monetization, invite investments, and receive fair value calculations for these assets. ■

34. Rules of Thumb: Valuing Patents and Technologies, *les Nouvelles*, December 1999.

35. Ibid.

36. Ibid.

37. Leveraging Software via the Capital Markets, *les Nouvelles*, September 2008).

38. *What is Your Software Worth?* Gio Wiederhold, Stanford University, April 2007.