Software Estimating Models: Three Viewpoints

Dr. Randall W. Jensen
Software Technology Support Center

Lawrence H. Putnam Sr.
Quantitative Software Management, Inc.

William Roetzheim
Cost Xpert Group, Inc.

This article compares the approaches taken by three widely used models for software cost and schedule estimation. Each of the models is compared to a common framework of first-, second-, and third-order models to maintain consistency in the comparisons. The comparisons illustrate significant differences between the models, and show significant differences in the approaches used by each of the model classes.

The purpose of this article is to present an unbiased comparison of three approaches to estimating software development costs. Rather than a single author attempting to arrive at a middle of the road, politically correct description of the three approaches, this article presents the comparison according to three individuals who are at the heart of these major estimating philosophies (from the horses' mouths so to speak).

Origins and Evolution of Software Models

This article prompted an enlightening trip back into the fuzzy history of computer-based software cost and schedule estimating methods and tools. It appears that the origins are not that remote, and the methods appeared over a relatively short period of time and have evolved in spurts and starts since then. The first real contribution to the estimating technology happened in 1958 with the introduction of the Norden staffing profile [1]. This profile has been incorporated in many of the estimating methodologies introduced since then.

A flurry of software estimating methods was introduced beginning in the mid-1970s and throughout the next decade. The first publication of any significance was presented by Ray Wolverton [2] of TRW in 1974. Wolverton was a major contributor to the development of the Constructive Cost Model (COCOMO) [3]. The second major contribution to the evolution of software estimating tools was the Doty Associates model [4], developed for the U.S. Air Force in 1977. The period from 1974 through 1981 brought most of the software estimating models (tools) we use today to the marketplace.

Each of these tools evolved at a gentle pace (refined algorithms and drivers) until about 1995, when significant changes were made to many of the models. COCOMO II, for example, had several releases between 1995 and 1999. Sage, released in 1995, is a major redefinition of the 1979 Seer model. Cost Xpert, introduced in 1996, is a major modification of the COCOMO family line. It is amazing that, in the 25 years elapsed since the first wave of estimating tools, development environments have changed so little that these models and their predicted environments are still valid today.

Framework for Discussion

When we look at software estimating models, they generally fall into one of three classes: first-, second- or third-order forms. This article compares three widely used models using the three classes as a framework for discussion and comparison.

First-Order Model

The first-order model is the most rudimentary model class. The model is simply a productivity constant, defining the production capability of the development organization in terms of arbitrary production units multiplied by the software product effective size to obtain the development effort or cost. The production units can be source lines of code (SLOC), function points (FPs), object points, use cases, and a host of other units. For the purpose of this discussion, we will use effective source lines of code (ESLOC) as the production measure, and person-hours per ESLOC as the productivity measure. This can be stated as follows:

$$E_d = C_k S_e \quad (1)$$

where,

E_d is the development effort in person hours (ph).
C_k is a productivity factor (ph/esloc).
S_e is the number of ESLOC.

The productivity factor is commonly determined by the product type, historic developer capability, or both as derived from past projects. As simple as this equation is, it is widely used to produce high-level, rough estimates. An expansion of this form used as far back as the 1970s is:

$$E_d = C_k(S_{new} + 0.75S_{modified} + 0.2S_{reused})\ ph \quad (2)$$

Or it is a similar variation. The weakness of this model is its insensitivity to the magnitude of the effective product size. Productivity is, or at least should be, decreased for larger projects.

Second-Order Model

The second-order model compensates for the productivity decrease in larger projects by incorporating an *entropy* factor to account for the productivity change. The entropy effect demonstrates the impact of a large number of communications paths that are present in large development teams. The number of paths is specified by n(n-1)/2 where n is the number of development personnel. The second-order model becomes the following:

$$E_d = C_k S_e^{ß} \quad (3)$$

where,

ß is an entropy factor that accounts for the productivity change as a function of effective product size.

An entropy factor value of 1.0 represents no productivity change with size. An entropy value of less than 1.0 shows a productivity increase with size, and a value greater than 1.0 represents a productivity decrease with size. Entropy values of less than 1.0 are inconsistent with historical software data[1]. Most of the widely used models in the 1980s (COCOMO embedded mode, PRICE-S, REVIC, Seer, and SLIM) used entropy values of approximately 1.2 for Department of Defense projects.

The major weakness of this model is

its inability to adjust the productivity factor to account for variations between projects in development environments. For example, contractor A may have a more efficient process than contractor B; however, contractor A may be using a development team with less experience than used in the historic productivity factor. Different constraints may be present in the current development than was present in previous projects. In addition, using a fixed, or calibrated, productivity factor limits the model's application across a wide variety of environments.

Third-Order Model

The third-order model compensates for the second-order model's narrow applicability by incorporating a set of environment factors to adjust the productivity factor to fit a larger range of problems. The form of this model is as follows:

$$E_d = C_k \left(\prod_{i=1}^{n} f_i\right) S_e^{\beta} \qquad (4)$$

where,

f_i is the ith environment factor.
n is the number of environment factors.

The number of environment factors varies across estimating models, and is typically between 15 and 32.

Using the generalized model set defined in equations (1) through (4), we have a framework for comparing the definition and features of the software estimating models. Three model types are described and compared in the following sections of this article: (1) models evolving from the 1979 Seer model developed and described by Dr. Randall Jensen, (2) models evolving from the COCOMO model described by William Roetzheim, and (3) the SLIM model developed and described by Lawrence Putnam, Sr.

Sage/Seer Effort and Schedule Calculations

Software development involves three important elements: people, a process, and a product. The people element describes management approach and style as well as personnel attributes, including capability, motivation, and communication effectiveness. Process represents the software development approach, tools, practices, and life-cycle definition. The product attributes include project-imposed constraints such as development standard, memory and time constraints, and security issues.

Software development is the most communication-intensive of all engineering processes. This unique software process characteristic suggests significant productivity gains are more likely to be realized through communication improvement, rather than through technology. Communication effectiveness is determined by organizational structure, management approach, and development environment. The Jensen model [5], and its implementations, embodies the impacts of the three important elements in software development cost and schedule estimates.

This section of the article discusses the underlying theory of a line of software cost and schedule estimating tools that evolved from the Jensen software model [6] at Hughes Aircraft Company's Space and Communications Group in 1979. The original model implementation was called Seer (not an acronym), a name later converted to the acronym SEER-SEM (Software Evaluation and Estimation of Resources-Software Estimating Model) [7] and trademarked by Galorath Associates, Inc. (GAI) in 1990. The Seer concepts were influenced by Lawrence Putnam's work [8] published in 1976 and the Doty Associates [9] estimating model published in 1977. The Seer model was derived from the U.S. Army data used by Putnam to develop SLIM, but with a different interpretation of the data itself.

The first major update to the Jensen model came in 1995 (Jensen II) with the addition of project management and personnel characteristics effects to the model. The impacts of motivation, management style, and teaming on productivity have long been suspected, but until 1995 the data to support the alleged behavior was simply insufficient to make credible model changes. These changes were implemented in the Software Engineering, Inc., Sage [10] software estimating system. For the sake of brevity, the Jensen model (I and II) will be referred to as Sage throughout this discussion. The following discussion applies to all descendants of the original Jensen estimating model. The fundamental equations are the following:

$$S_e = C_{te} \sqrt{K} T_d \qquad (5)$$

and

$$D = \frac{K}{T_d^3} \qquad (6)$$

where,

C_{te} is the effective technology constant of the development activity.
K is the total life-cycle effort in person-years (py) of the software development starting with the software requirements review through the software's end of life.
T_d is the software product development time in years.
D is the product complexity rating.

Equations (5) and (6) solved simultaneously provide both schedule and effort estimates in one calculation with the relationship between effort and schedule linked by the product complexity. This approach is unique to the Jensen and Putnam models.

The parameter D is the same as the *Difficulty* parameter discovered by Putnam. This explains the use of D to describe what the Jensen model refers to as complexity.

Development effort is defined by

$$E_d = 0.3945K$$

where,

E_d is the development effort in py through the final qualification test.

The Sage software development effort equation is an implementation of the third-order model discussed in the introduction to the model comparison even though it is not immediately apparent. Combining equations (5) and (6), we find the following:

$$E_d = \frac{18{,}797 D^{0.4}}{C_{te}^{1.2}} S_e^{1.2} \text{ person months (pm)} \qquad (7)$$

The ugly part of equation (7) preceding the effective size element comprises the productivity factor of the third order model. The effective technology constant C_{te} contains two components: (1) the basic technology constant C_{tb} representing the development organization's raw capability; that is, capability independent of the constraints imposed by a specific develop-

Table 1: *Sage Analyst Capability Ratings*

Definition	Highly motivated AND experienced team organization	Highly motivated OR experienced team organization	Traditional software development organization	Poorly motivated OR non-associative organization	Poorly motivated AND non-associative organization
Relative cost impact	0.71	0.86	1.00	1.19	1.46

ment, and (2) the impacts of 24 environment factors/constraints. The C_{te} value is obtained from the following:

$$C_{te} = \frac{C_{tb}}{\prod_{i=1}^{24} f_i} \quad (8)$$

where,

C_{tb} represents the basic technology constant.
f_i is the ith product-impacted environment factor.

The C_{tb} value can be anywhere between 2,000 and 20,000 with a normal range between 5,500 and 7,500. The highest value observed from available data at this time is about 8,635. Higher values obviously imply higher productivity and efficiency. Theoretical values of C_{te} range from 0 through 20,000. The practical upper C_{te} limit is defined by an organization's rating. The practical lower C_{te} bound is about 500 for a less-than-average organization and severe product constraints.

The relative cost impact of the analyst capability rating for Sage is shown in Table 1 as an example of one of the 24 environment factors.

The product development time T_d is the minimum development time as can be seen from the Paul Masson cost-time relationship shown in Figure 1. The *Paul Masson Point* represents the minimum development time that can be achieved with a specified size, environment, and complexity. By attempting to reduce the schedule below the minimum time, the cost will increase and the schedule will also increase as described by Brooks Law [11]: "Adding people to a late software project makes it later." Sage computes the minimum development schedule as a default.

The region shown by the double-headed arrow is represented as a square-law relationship in the model between cost and schedule, or $c = KT_d^2$. At first glance it seems that a longer schedule should equate to higher cost. By explaining the phenomenon in two logical steps, the productivity gain over the region becomes clear. A longer schedule requires a smaller development team. A smaller team is more efficient; thus, productivity improves and the cost decreases. This phenomenon applies until the productivity gain is eaten up by fixed costs.

As an example of the effort and schedule predicted by Sage, let us assume the following development project parameters: The product is a satellite mission operations system (application with significant logical complexity with some changes to the underlying operating system) consisting of 59,400 new SLOC. A basic technology rating of 7,606 places the developer at the upper end of the typical capability range. The project environment constraints, including the Institute of Electrical and Electronics Engineers Standard 12207 development standard, reduce the effective technology rating to 2,624. The third-order equation form of equation (7) reduces to $E_d = 4.007S_e^{1.2}$. The results are tabulated in Table 2.

The minimum development schedule, which is simultaneously calculated in equation (5), is approximately 25 months.

Quantitative Software Management View of Software Estimating and Productivity Measurement

Productivity measurement is used to:

1. Tune estimating systems.
2. Baseline and measure progress in software development.

But what is *productivity*? My answer is: It is not SLOC/PM or FP/PM. This is the traditional view from economic theory – output/input.

The software industry has 35 years of experience that shows that this ratio works poorly as a productivity metric. At Quantitative Software Management (QSM), we have learned why it does not work: because it ignores schedule, and software development is very sensitive to schedule. A vast amount of our 27-year collection of data, some 6,600 completed systems, coupled with empirical analysis shows that schedule is the most important factor in estimating relationships and must be dealt with explicitly. If schedule is not so recognized and dealt with, then it will assert itself implicitly and cause much grief. By this, I mean both time and effort must be included multiplicatively in an expression for a good software algorithm. We have found this to be of the conceptual form:

Amount of function = Effort * Schedule * Process Productivity (9)

where,

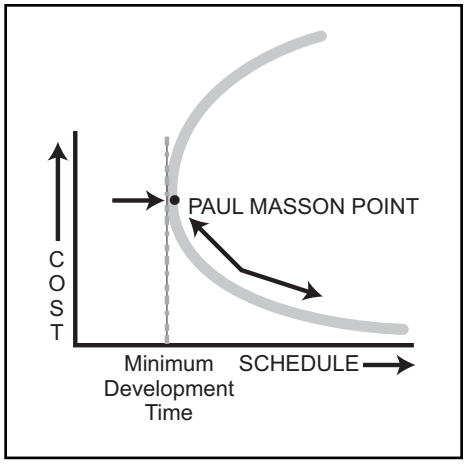


Figure 1: *Paul Masson Rule*

Effort and Schedule have exponents. Specifically,

Size = (Effort/ßeta)$^{1/3}$ Schedule$^{4/3}$ Process Productivity Parameter (10)

where,

***Size* is the size in SLOC, or other measure of amount of function.**
***Effort* is the amount of development effort in py.**
***ßeta* is a special skills factor that varies as a function of size from 0.16 to 0.39.**
***ßeta* has the effect of reducing process productivity, for estimating purposes, as the need for integration, testing, quality assurance, documentation, and management skills grows with increased complexity resulting from the increase in size.**
***Schedule* is the development time in years.**
***Process Productivity Parameter* is the productivity number that we use to tune the model to the capability of the organization and the difficulty of the application. We do this by calibration as explained in the next section. The theoretical range of values is from 610 to 1,346,269. The range of values seen in practice across all application types is 1,974 to 121,393 and varies exponentially.**

Estimating and Tuning Models

All models need tuning to moderate the *productivity* adjusting factor.

- Some models use effort multipliers or modifiers.
- I have found that we can use the software equation to *calibrate* our estimat-

Table 2: *Example Estimate*

S_e kesloc	D	C_{tb}	C_{te}	E_d . pm	T_d mo	Productivity, sloc/pm
59.4	12	7,606	2,624	538.7	25	110

ing algorithm. This calibration process is far more accurate than intelligent guesses of the settings for effort multipliers because it is based on real data from the development organization. All we need to do is to rearrange the software equation into this form:

$$\text{Process Productivity Parameter} = \text{Size} / ((\text{Effort}/\beta\text{eta})^{1/3} (\text{Schedule}^{4/3})) \quad (11)$$

From historic projects we know the size (SLOC, FPs, etc.), effort (py) and schedule (years). Then just put in a consistent set of historic numbers and we can calculate a Process Productivity Parameter. This works well. Note that the expression for Process Productivity Parameter includes schedule and that it is multiplicatively tied to effort. This expression is our definition of software productivity.

This software equation has two variables that we want to solve for: schedule and effort. That means we have to have another equation to get a solution in the form of a schedule-effort pair. The second equation may be in the form of a constraint like maximum budget for the project (Maximum Development Effort = Maximum Cost/$Average Burdened Labor Rate), or, Maximum Schedule = Maximum Development Time in years. There are a number of other constraints that can be used such as peak manpower, or maximum manpower buildup rate (defined as *Difficulty* in Randall Jensen's preceding section).

Example of an Estimate

Here is a simple example: We need an estimate for a Global Positioning System navigation system for an air-launched, land-attack missile. We have completed the high-level design phase. Estimated size is 40,000 C++ SLOC; Process Productivity Parameter for this class of work (real time avionic system) is 3,194 [taken from Table 14.8, in 12], *ßeta* = 0.34 [taken from Table 14.4, in 13]. We have to deliver the system to the customer for operational service in two years (24 months from the end of high-level design) The fully burdened contractor labor rate is $200,000 per person-year. Substituting in the software equation and solving for effort, we have the following:

$$40{,}000 = (\text{Effort}/0.34)^{1/3}\, 2^{4/3}\, 3{,}194$$
$$\text{Effort} = 41.74 \text{ PY (Approximately 500.9 pm)}$$
$$\text{Cost} = \$200{,}000/\text{PY} * 41.74 \text{ PY} = \$8.35 \text{ million}$$

Observations Concerning Effort Multipliers/Moderators

Many estimating systems use a family of adjusting factors to try to tune their productivity constant. Between 15 and 25 of these *tweekers* are typical. The values are picked from a scale centered on 1.0 that increase or decrease the Productivity constant. This process does moderate the baseline productivity value. Unfortunately, it is highly subjective – dependent upon the judgment of the practitioner. It is not consistent or reproducible from person to person and hence it introduces considerable uncertainly into the estimate that follows.

At QSM, we use a family of tweekers for tools and methods, technical complexity of the project, competence, experience, and skill of the development team. This list is similar to most other estimating systems. But we use it only as a *secondary* technique for those organizations that truly have no historic data. The notion of calibration from historic data is much more accurate and powerful because it captures the real capability and character of the development organization in a single number – the Process Productivity Parameter in the software equation. This single number is unambiguous and consistent; there is no subjectivity involved.

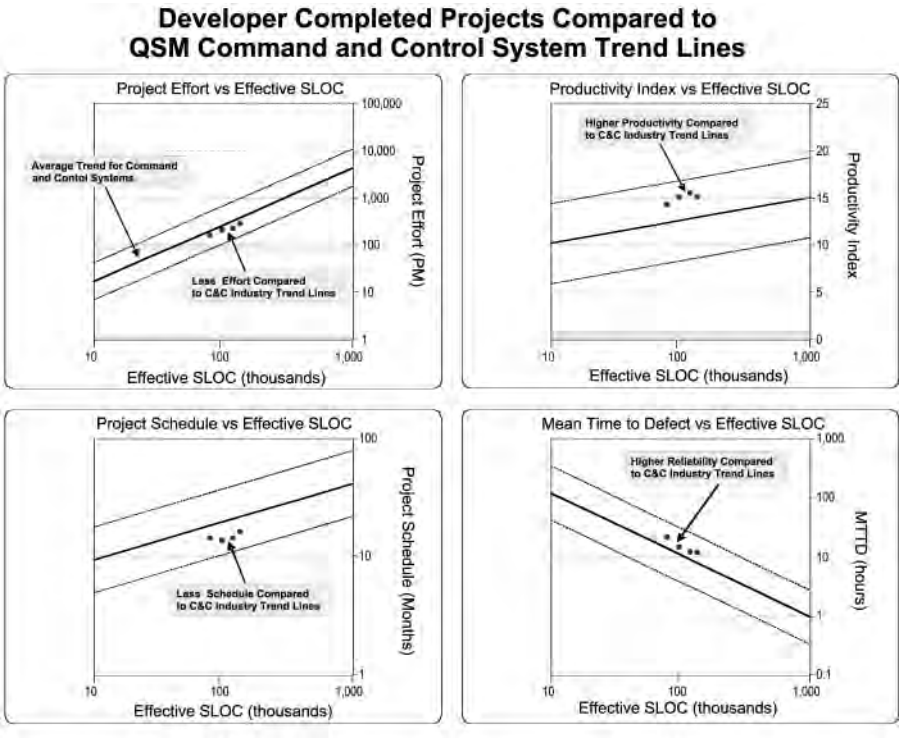
Benchmarking

One of the nice things about being able to substitute your historical data into the software equation is that you can calculate an unambiguous number that can be used for benchmarking. We transform the Process Productivity into a linear scale and call that a Productivity Index (PI). If we collect a homogeneous set of data from a development organization and calculate the PI for each project, we see a fairly normal distribution with a central tendency.

The central tendency represents our current average PI. If we keep track of all our projects over time and if we are doing process improvement (trying to move up the SEI scale) then we will see an increase in the PI over time. Often we can plot the PI behavior over time and pick up the trend. Moreover, this graphical approach makes it easy to compare organizations doing similar types of work. Extending this thinking a little bit provides the ability to quantitatively compare the real capability of bidders on a software development contract. This comparison process takes a lot of guesswork out of trying to determine the real capability of vendors.

The benchmarking idea can be extended easily to show performance capabilities of the development organization. The idea is to take a fairly large body of contemporaneous historic data from the same industry sector, then generate some trend lines plots of the generic form: management metric (schedule, effort, staffing, defects, etc.) versus size (SLOC, FPs, objects, etc.). Next, superimpose data points from the development

Figure 2: *Trend Lines Plots*



organization being measured on top of these trend lines and see how they position (high or low) compared with the industry average trend line at the appropriate size. For example, if your data shows a pattern of falling below the average line for effort, schedule, and defects you are a more effective producer (high productivity developer). Almost invariably your PI and Mean Time to Defect (MTTD) will be higher as well. This means you can unambiguously and quantitatively measure productivity. After nearly 30 years of experience doing it, we know it works consistently and well. An example of such plots is shown in Figure 2.

Cost Xpert Effort and Schedule Calculations

Algorithmically, Cost Xpert started with the COCOMO models, added Revised Intermediate COCOMO (REVIC) extensions, and then layered functionality on top of this base. The core approach of Cost Xpert is as follows:

1. Define scope using various measures of size (e.g., SLOC, FPs, class-method points, user stories, many others) for new development, reused code, and commercial off-the-shelf (COTS) components).
2. Use scope and a variety of adjusting factors (addressed below) to calculate effort.
3. Use effort to calculate optimal schedule.
4. Feedback deviations from the optimal schedule to adjust effort, if necessary.

First-Order Modeling in Cost Xpert

First order modeling involves linear calculations of effort using a productivity constant:

$$\text{Effort} = \text{Productivity} * \text{Size} \qquad (12)$$

In Cost Xpert, all sizing metrics (SLOC, FPs, class-method points, user stories, etc.) and development stage (new, reused, COTS) are normalized to a SLOC equivalent. This SLOC equivalent is defined such that it is valid for estimating effort, although in some situations it may not accurately represent physical lines of code (for example, in environments where much of the code is auto-generated). In other words, although it once represented physical lines of code calculated using an approach called backfiring, and although that relationship is still true for older development environments, with newer environments it has become more of an estimating size proxy.

COCOMO and REVIC use(d) relatively small databases of projects and have a correspondingly small universe of productivity numbers. For example, COCOMO II uses the value 2.94 person-months per thousand SLOC (KSLOC) [14]. Commercial vendors, including the Cost Xpert Group, are able to maintain much larger databases of projects and hence can segment those databases into a finer granularity of project classes, each with a corresponding productivity number.

Cost Xpert uses a project type variable to denote the nature of the project (e.g., military, commercial, and internet) and set the coefficient for given historic database segments. Actual numbers for these sample project classes are shown in Table 3 [15]. The productivity number multiplied by KSLOC yields the first order effort in person-months.

However, Cost Xpert Group research has determined that there are additional sources of variation across projects and project domains (see the Third-Order Modeling section for examples). Our calibrations account for the additional sources of variation and therefore produce different coefficients for given project classes than the models with reduced factor sets. The overall net productivity in Cost Xpert thus accounts for second-order and third-order modeling described in the following sections.

Second-Order Modeling in Cost Xpert

Second-order modeling in Cost Xpert involves adjusting the productivity to allow for economies or diseconomies of scale. An economy of scale indicates that the productivity goes up as the scope goes up. For example, you would expect that it is cheaper per yard to install 100,000 yards of carpeting than 1,000 yards. A diseconomy of scale indicates that the productivity goes down as the scope goes up. In software, we are dealing with diseconomies (larger projects are less efficient). This is modeled by raising the size to a power, with a power greater than 1.0 increasing the apparent scope, and thereby effort, with increasing project size. The second-order model looks like this:

$$\text{Effort} = \text{Productivity Factor} * \text{Size}^{\text{Scaling Factor}} \qquad (13)$$

Table 4 shows some scaling factors by project type.

Third-Order Modeling in Cost Xpert

Third-order modeling in Cost Xpert involves adjusting the productivity and scaling factors by project-specific variables. Cost Xpert uses 32 variables, called environment variables (E) to adjust the productivity number up or down, and five variables, called scaling variables (S) to adjust the scaling factor up or down. Each variable is set to a value ranging from very low to extremely high using defined criteria for each setting. Many of these variables will typically be fixed at a given value for an organization, with only a handful actually varying from project to project. The total productivity factor adjustment (PFA) is the product of the individual productivity factor values.

Project Type	Productivity Factor (pm/KSLOC)
Military	3.97
Commercial	2.40
Internet	2.51

Table 3: *Productivity Factors*

Project Type	Scaling Factor
Military, Complex	1.197
Military, Average	1.120
Military, Simple	1.054
Commercial	1.054

Table 4: *Scaling Factors for Various Project Types*

$$\text{PFA} = \prod \text{E} \qquad (14)$$

Table 5 (see page 28) shows some sample environmental variables and the resultant PFAs as a sample of how this works.

The higher the number, the more effort required to deliver the same KSLOC, so in the Table 5 sample the commercial product would require less effort per KSLOC than the military project.

Additionally, Cost Xpert has introduced different types of linear factors for important sources of effort variation. We find that much of the variance between project classes (e.g., military versus commercial) can be accounted for by the life cycles and standards typically employed. We have introduced the following factors:

- **Project Life Cycle:** The life cycle (template of activities, or work breakdown structure) used for development).
- **Project Standard:** The deliverables (engineering specifications, or artifacts) produced during development.

These factors are rated by selecting named life cycles and standards, which are different than rating the standard environmental variables. In addition to adjusting

	Required Reliability (E)		Multi-Site Development (E)		Security Classification (E)		Net Productivity (PFA)
Military	Very High	1.26	Nominal	1.00	High	1.10	1.386
Commercial	Nominal	1.00	Very High	0.86	Nominal	1.00	0.86

Table 5: *Sample Productivity Factor Adjustments*

	Life Cycle and Multiplier		Standard and Multiplier		Net Productivity
Sample 1	Waterfall	1.01	Military-498	1.34	1.35
Sample 2	RAD	0.91	RAD	0.51	0.46

Table 6: *Life Cycle and Standard Adjustment Factors*

effort, the life cycle and standard are used to create specific detailed project plans and page size estimates. They are also used in our defect model. Examples using actual numbers for sample projects are shown in Table 6, where RAD is rapid application design. These two samples could represent military and commercial projects respectively.

The relative productivity difference between the samples due to these two factors would be 1.35/.46 = 2.9 or 290%.

The five scaling variables (S) (not shown) work in a somewhat analogous manner, but the five factors are summed to adjust the exponential factor that applies to the diseconomy of scale. The total third-order formula is then the following:

$$\text{Effort} = \prod E * P * \text{KSLOC}^{\text{Scaling Factor} + (\sum s/5)} \quad (15)$$

where,

Effort **is the effort in pm.**
E **are the various environmental factors.**
P **is the productivity factor (which is further broken down into a project type, life cycle, and standard).**
KSLOC **is the SLOC equivalent in thousands.**
S **are the five scaling factor adjustments.**
ScaleFactor **is the default scaling factor for this project type.**

If we use the military productivity factor from Table 3, military-complex from Table 4, the military PF adjustment from Table 5, and the life cycle and standard adjustments for sample 1 in Table 6, the equation simplifies to:

$$\text{Effort} = 3.97 * 1.386 * 1.35 * \text{KSLOC}^{(1.197 + (\sum s/5))} \quad (16)$$
$$\text{Effort} = 7.43 * \text{KSLOC}^{(1.197 + (\sum s/5))}$$

Calculating Schedule and Adjusting for Schedule Deviations

In Cost Xpert, the optimal schedule is driven by the calculated effort. The schedule formula is of the form:

$$\text{Schedule} = \propto * \text{Effort}^{\beta} \quad (17)$$

where,

Schedule **is the schedule in calendar months.**
∝ is a linear constant.
ß **is an exponential constant.**

Table 7 shows a couple of sample values.

Accelerating a project from this calculated schedule results in inefficiencies that then lower productivity and increase effort. Cost Xpert handles this schedule acceleration adjustment to productivity through a table lookup and interpolation between points in the table.

Table 7: *Sample Schedule Factors*

Project Type	α	ß
Military	3.80	0.378
Commerical	2.50	0.3348

Summary and Conclusions

The three model implementations described in this article represent the majority of the estimating approaches available to the estimator today. Our intent in this article is not to advocate a single software estimating approach or tool, but is to expose you, the reader, to the mindsets incorporated in the more widely used approaches available today.◆

References

1. Norden, P.V. "Curve Fitting for a Model of Applied Research and Development Scheduling." IBM Journal of Research and Development 2.3 (July 1958).
2. Wolverton, R.W. "The Cost of Developing Large-Scale Software." IEEE Transactions on Computers June 1974: 615-636.
3. Boehm, B.W. Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
4. Herd, J.R., J.N. Postak, We. E. Russell, and K.R. Stewart. "Software Cost Estimation Study – Final Technical Report." Vol. I. RADC-TR-77-220. Rockville, MD: Doty Associates, Inc., June 1977.
5. Jensen, R.W. "Management Impact of Software Cost and Schedule." CROSSTALK July, 1996:6.
6. Jensen, R.W. A Macrolevel Software Development Cost Estimation Methodology. Proc. of the Fourteenth Asilomar Conference on Circuits, Systems and Computers. Pacific Grove, CA, 17-19 Nov. 1980.
7. Galorath, Inc. SEER-SEM Users Manual. El Segundo, CA: Galorath Inc., Mar. 2001.
8. Putnam, L.H. A Macro-Estimating Methodology for Software Development. Proc. of IEEE COMPCON '76 Fall, Sept. 1976: 138-143.
9. Herd, J.R., J.N. Postak, We. E. Russell, and K.R. Stewart. "Software Cost Estimation Study – Final Technical Report." Vol. I. RADC-TR-77-220. Rockville, MD: Doty Associates, Inc., June 1977.
10. Software Engineering, Inc. Sage User's Manual. Brigham City, UT: Software Engineering, Inc., 1995.
11. Brooks Jr., F.P. The Mythical Man-Month. Reading, MA: Addison-Wesley, 1975.
12. Putnam, Lawrence H., and Ware Myers. Measures for Excellence. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992: 237
13. Putnam. Measures for Excellence, 234.
14. Boehm B., et al. Software Cost Estimation With COCOMO II. Prentice-Hall, 2000.
15. Cost Xpert Group, Inc. Cost Xpert 3.3 Users Manual. San Diego, CA: Cost Xpert Group, Inc., 2003.

Note

1. This is a good place to point out a major difference between software and almost any other manufactured product. In other estimating areas, a large number of products improves productivity through the ability to spread costs over a large sample and reduce learning curve effects. The software product is but a single production item that becomes more complex to manage and develop as the effective size increases.

About the Authors

Randall W. Jensen, Ph.D., is a consultant for the Software Technology Support Center, Hill Air Force Base, with more than 40 years of practical experience as a computer professional in hardware and software development. He developed the model that underlies the Sage and the Galorath, Inc. SEER-SEM software cost and schedule estimating systems. He retired as chief scientist in the Software Engineering Division of Hughes Aircraft Company's Ground Systems Group. Jensen founded Software Engineering, Inc., a software management-consulting firm in 1980. Jensen received the International Society of Parametric Analysts Freiman Award for Outstanding Contributions to Parametric Estimating in 1984. He has published several computer-related texts, including "Software Engineering," and numerous software and hardware analysis papers. He has a Bachelor of Science, a Master of Science, and a doctorate all in electrical engineering from Utah State University.

Software Technology Support Center
6022 Fir AVE BLDG 1238
Hill AFB, UT 84056
Phone: (801) 775-5742
Fax: (801) 777-8069
E-mail: randall.jensen@hill.af.mil

Lawrence H. Putnam Sr. is the founder and chief executive officer of Quantitative Software Management, Inc., a developer of commercial software estimating, benchmarking, and control tools known under the trademark SLIM. He served 26 years on active duty in the U.S. Army and retired as a colonel. Putnam has been deeply involved in the quantitative aspects of software management for the past 30 years. He is a member of Sigma Xi, Association for Computing Machinery, Institute of Electrical and Electronics Engineers (IEEE), and IEEE Computer Society. He was presented the Freiman Award for outstanding work in parametric modeling by the International Society of Parametric Analysts. He is the co-author of five books on software estimating, control, and benchmarking. Putnam has a Bachelor of Science from the United States Military Academy and a Master of Science in physics from the Naval Postgraduate School.

Quantitative Software Management, Inc.
2000 Corporate Ridge STE 900
McLean, VA 22102
Phone: (703) 790-0055
Fax: (703) 749-3795
E-mail: larry_putnam_sr@qsm.com

William Roetzheim is the founder of the Cost Xpert Group, Inc., a Jamul-based organization specializing in software cost estimation tools, training, processes, and consulting. He has 25 years experience in the software industry, is the author of 15 software related books, and over 100 technical articles.

2990 Jamacha RD STE 250
Rancho San Diego, CA 92019
Phone: (619) 917-4917
Fax: (619) 374-7311
E-mail: william@costXpert.com

Letter to the Editor

Dear CrossTalk Editor,

In the December 2005 issue, the article "Agile Software Development for the Entire Project" by Granville Miller, Microsoft, describes how the agile process in MSF can make the *agile* described in the Agile Manifesto <www.agilemanifesto.org> much easier to implement without all of those difficult changes that many others have experienced. He describes how these reflect the fine engineering practices at Microsoft that have led the MSF version of agile to already be a year late.

It has taken more than 20 years for parts of the American manufacturing industry to adopt lean thinking. Agile, which has many parallels to lean manufacturing, will also take a lot of effort and time. Change is always an effort, and only the dramatic benefits of agile make it worthwhile. Efforts by people like Granville Miller to water down agile by redefining the intent do not help. Efforts that add more process miss the point; process is defined by self-managing teams within frameworks. Decisions are made by these teams working closely with customers to maximize benefit and optimize results.

At the start of the agile movement, we were warned that the larger commercial interests would attempt to water it down to *fit* their existing tools. We should expect to see other similar fits such as from IBM (through RUP in the Eclipse Foundation) and others. The *refinements* suggested by Granville Miller do a disservice to everyone working on agile.

Ken Schwaber
Signatory to the Agile Manifesto
Founder of the Agile Alliance
Co-Author of the Scrum Agile process
ken.schwaber@controlchaos.com