Presented at the 2007 ISPA/SCEA Joint Annual International Conference and Workshop - www.iceaaonline.com

A New Software Estimating Framework: 25 Years and Thousands of Projects Later

Mike Ross
President & CEO
r2Estimating, LLC
7755 E. Evening Glow Dr.
Scottsdale, AZ 85262-1295
480-488-8366
mike.ross@r2estimating.com
http://www.r2estimating.com

[1,2]***Abstract***—It's about time we in the software development community revisit the assumptions, relationships, and flexibility contained in our currently-available software estimating models. Most of the current models still implement fundamental relationships that are based on at least 25 years old data and assumptions. In the meantime, data from many thousands of projects have since been collected and offer an opportunity to revisit old assumptions and relationships. This paper documents the basis, assumptions, and derivations behind a set of general software effort, duration, and defects estimating relationships that are based on the notion that software construction is the cumulative effect of people doing work (effort) over some duration (period of elapsed calendar time) that produces a desired software product (size) and unwanted byproducts (defects). This set of relationships is derived from several evidently-good correlations, the primary three being: 1) effort generally increases with increasing size, 2) duration generally increases with increasing effort, and 3) effort generally increases with increasing defects. This derivation ultimately yields three limited tradeoff relationships: one between effort and duration, one between cost and duration, and one between defects and duration.

TABLE OF CONTENTS

---

[1] © 2007 r2Estimating, LLC. All rights reserved.
[2] Rev D, April 9, 2007

1. INTRODUCTION

Purpose

The purpose of this paper is to document the basis, assumptions, and derivations behind a set of general software effort, duration, and defects estimating relationships that are based on the notion that software construction is the cumulative effect of people doing work (effort) over some duration (period of elapsed calendar time) that produces a desired software product (size) and undesired byproducts (defects).

Scope

The derivations, assumptions, and resulting model described by this paper establish relationships between size, efficiency, defect vulnerability, effort, duration, and defects and are henceforth collectively referred to as the ***r2 Software Estimating Framework (r2SEF)***™ and are implemented in the *r2ESTIMATOR*™ estimating tool[3].

Background

Most of our current software estimating models still implement fundamental relationships that are based on at least 25 years old data and assumptions. In the meantime, data from many thousands of projects have since been collected and offer an opportunity to revisit old assumptions and relationships. To claim that this new data continues to "re-validate" these old assumptions and relationships is to claim that software development as a process is static and is to ignore its evolution with respect to management techniques, team behavior, host and target platforms, development methodologies, functionality abstraction, etc. In fact, analysis of this new data against these old relationships suggests that the underlying old assumptions are inappropriately restrictive. There are aspects of the old models that are held constant; the new data suggesting that these aspects should be variable according to the particular organization proposing to do the work and the type of project being proposed; *i.e., one size does not necessarily fit all*.

Current Model Types

To facilitate discussion of software estimating models, we suggest the following categorization scheme which was inspired by Jensen[4], introduced by Ross[5], and updated to accommodate the findings of this paper.

Type 0—Dart board, dice, roulette wheel, tarot cards, crystal ball, Ouija™ board.

Type 0.5—Engineering judgment.

---

[3] *r2ESTIMATOR*™ is developed and distributed by *r2ESTIMATING*®, *LLC*; http://www.r2estimating.com.

[4] This categorization was inspired by and is very similar to that found in [6]; the difference is this categorization's emphasis on the existence of an effort-time tradeoff relationship as a type criteria.

[5] [11] p. 2.

Type 1—A univariate relationship that assumes total construction effort E_C to be directly proportional to a linear function of effective software size S_e where the constant of proportionality ϖ represents construction average productivity[6] (effective software size per unit of effort) and the optional offset c represents fixed (i.e., size-independent) effort.

$$E_C = \frac{S_e}{\varpi} + c \tag{1}$$

Type 2—A pair of univariate independent power relationships: the first assumes total construction effort E_C to be directly proportional to a power function of effective software size S_e and the second assumes total construction duration t_C to be directly proportional to a power function of construction effort E_C where the constants of proportionality χ_1 and χ_2 represent complexity scale factors[7] (e.g., COCOMO [1], and derivatives).

$$E_C \propto f_1(S_e) \text{ and } t_C \propto f_2(E_C) \tag{2}$$

where

$$\begin{aligned} &f_1(x) = x^{a_1} \text{ and } f_2(x) = x^{a_2} \\ &\therefore E_C = \chi_1 S_e^{\,a_1} \text{ and } t_C = \chi_2 E_C^{\,a_2} \end{aligned} \tag{3}$$

Type 3—A bivariate power relationship that assumes construction average productivity $\varpi \equiv S_e / E_C$ is inversely proportional to a power function of the project's maximum staffing rate[8] $\dot{P}_{\max}$, the maximum staffing rate assumed to be $\dot{P}_{\max} \equiv K / t_C^{\,2}$ where K is total life cycle effort and is assumed to be $E_C / 0.3839$. The constant of proportionality c_k represents an efficiency scale factor[9] sometimes referred to as effective technology or process productivity (e.g., Jensen [5], Putnam [10], and derivatives).

---

[6] Source of the ubiquitous *lines per day* and *lines of code per person-month* metrics.

[7] COCOMO uses a system of units that measures effective software size in source lines of code (SLOC), effort in person-months, and duration in calendar months.

[8] This assumption is based on observations made by Norden [8] and elaborated for software development by Putnam [10] that suggest both construction and life cycle staffing follow the probability density function form of the Rayleigh distribution. This assumption implies, as can be seen by the function's first derivative (rate function), that the project's maximum staffing rate occurs at project start and is equal to life cycle effort divided by the square of construction duration.

[9] Jensen [5] and Putnam [10] both use a system of units that measures effective software size in source lines of code (SLOC), effort in person-years, and duration in calendar years.

$$\varpi \propto \frac{1}{f\left(\dot{P}_{\max}\right)} \text{ where } f\left(x\right) = x^a$$

$$\varpi = c_k \frac{1}{\dot{P}_{\max}^{\ a}} \tag{4}$$

$$\therefore E_C^{(1-a)} t_C^{\ 2a} = \frac{S_e}{0.3839^a c_k} \text{ or } E_C = \left(\frac{S_e}{0.3839^a c_k}\right)^{\left(\frac{1}{1-a}\right)} t_C^{\ -\left(\frac{2a}{1-a}\right)}$$

Type 4—A bivariate relationship that assumes the product of power expressions for both construction effort E_C and construction duration t_C to be proportional to a power expression for effective software size S_e where the reciprocal of the constant of proportionality is assumed to be an efficiency scale factor η and where k_η resolves the system of units being used; its value being unity when effort is measured in person-weeks and duration is measured in calendar weeks.

$$E_C^{\ \alpha_E} t_C^{\ \alpha_t} = \frac{S_e}{k_\eta \eta} \text{ or } E_C = \left(\frac{S_e}{k_\eta \eta}\right)^{\frac{1}{\alpha_E}} t_C^{\ -\left(\frac{\alpha_t}{\alpha_E}\right)} \tag{5}$$

The remainder of this paper describes the derivation of a Type 4 model hereinafter referred to as the ***r2 Software Estimating Framework (r2SEF)***.

Note that the Type 2 form can be converted to an instantiation of the Type 4 form by multiplicatively combining the two resultant equations in Equation (3).

$$E_C = \chi_1 \left(S_e\right)^{a_1} \text{ and } t_C = \chi_2 \left(E_C\right)^{a_2}$$

$$\therefore E_C^{\ \left(\frac{1-a_2}{a_1}\right)} t_C^{\ \left(\frac{1}{a_1}\right)} = \left(\chi_1 \chi_2\right)^{\left(\frac{1}{a_1}\right)} S_e \text{ or } E_C = \left(\frac{\left(\chi_1 \chi_2\right)^{\left(\frac{1}{a_1}\right)} S_e}{t_C^{\ \left(\frac{1}{a_1}\right)}}\right)^{\left(\frac{a_1}{1-a_2}\right)} \tag{6}$$

This Type 2 instantiation of Type 4 implies

$$\alpha_E = \frac{1-a_2}{a_1} \text{ and } \alpha_t = \frac{1}{a_1} \text{ and } \eta \propto \frac{1}{\left(\chi_1 \chi_2\right)^{\left(\frac{1}{a_1}\right)}} \tag{7}$$

Note also that the Type 3 form (resultant Equation (4)) can be viewed as an instantiation of the Type 4 form where the exponents on effort and duration are constrained by the Rayleigh-based assumption that construction average productivity is inversely proportional to a power function of the project's maximum staffing rate.

This Type 3 instantiation of Type 4 implies

$$\alpha_E = 1 - a \text{ and } \alpha_t = 2a \text{ and } \eta \propto 0.3839^a c_k \quad (8)$$

where for Jensen [5] $a \equiv 0.5$ and for Putnam [10] $a \equiv 0.\overline{6}$.

2. OBSERVATIONS AND HYPOTHESES

Fundamental Observations

Intuition, personal experience, and analysis of historical project data (effort, duration, size, and defects) using ordinary least squares (OLS) regression in log-log space yields the following observations:

- ***No free software***—Effort (and hence cost) increases monotonically as a function of increasing size (see Figure 1 below).

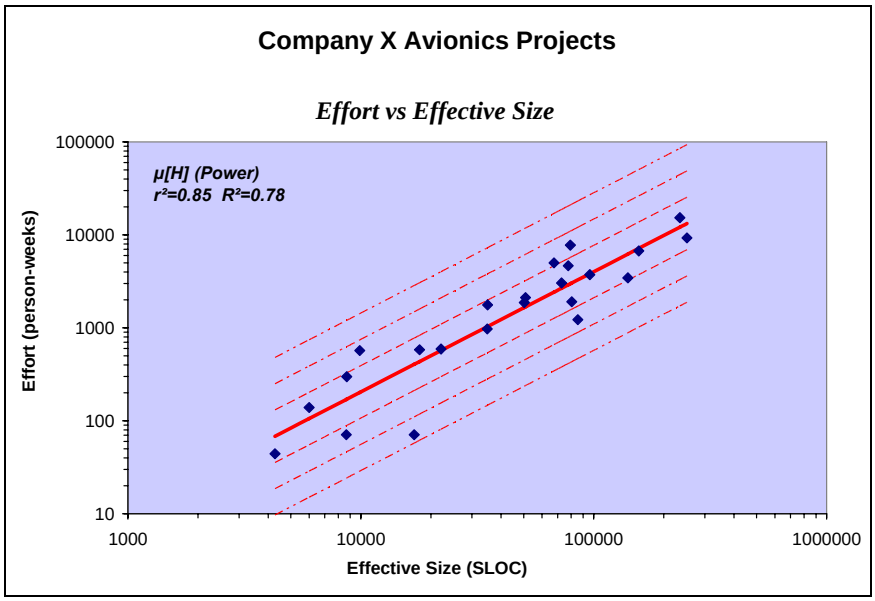

Figure 1. Effort Increases as Size Increases[10]

[10] r^2 is the square of the Pearson product-moment correlation coefficient between log(x) and log(y). R^2 is the square of the Pearson product-moment correlation coefficient between the actual y_i's and the estimated y_i's ($f(x_i)$'s), in this case assuming a y-intercept of zero. [3] p. 94.

- ***No instant software***—Duration increases monotonically as a function of increasing size (see Figure 2 below).

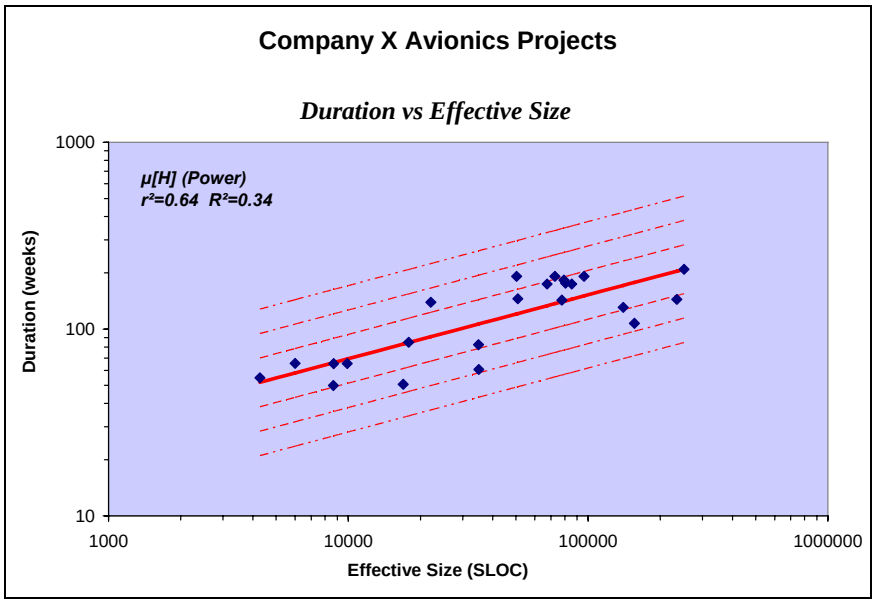

Figure 2. Duration Increases as Size Increases

- ***No perfect software***—Effort increases monotonically as a function of increasing defect count (see Figure 3 below).

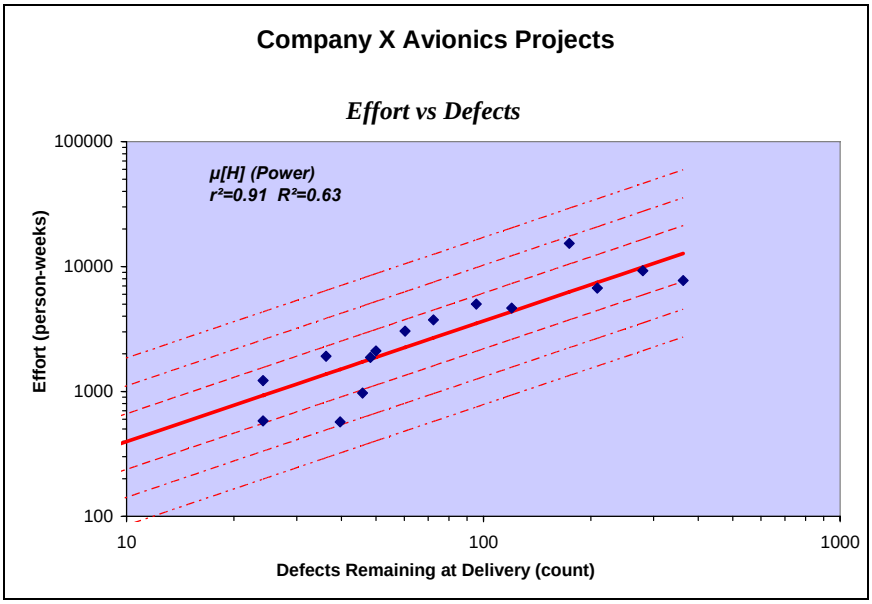

Figure 3. Effort Increases as Defects Increase

- ***Smaller teams are more productive*—**Productivity increases monotonically as a function of decreasing team size (see Figure 4 below).

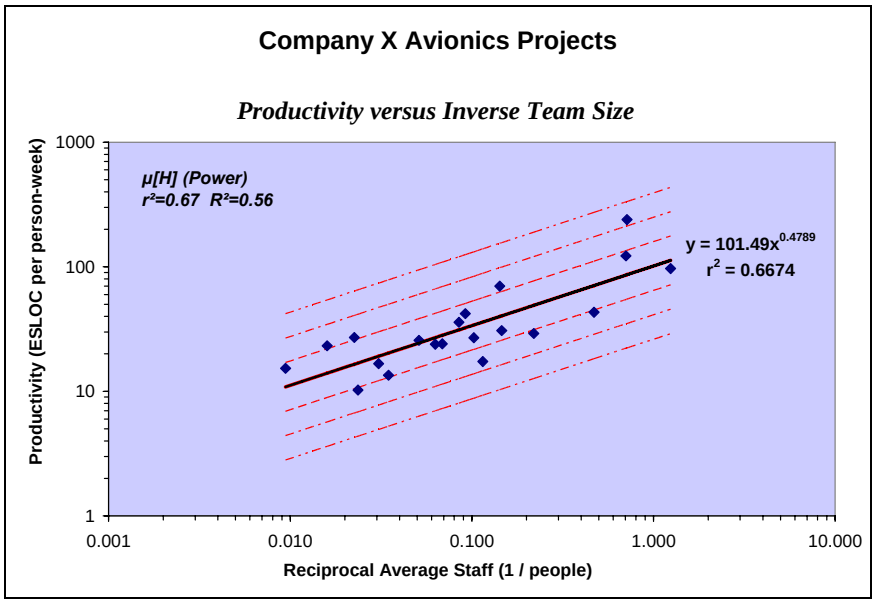

Figure 4. Productivity Increases as Team Size Decreases

- ***Smaller teams produce fewer defects*—**Defect count increases monotonically as a function of increasing team size (see Figure 5 below).

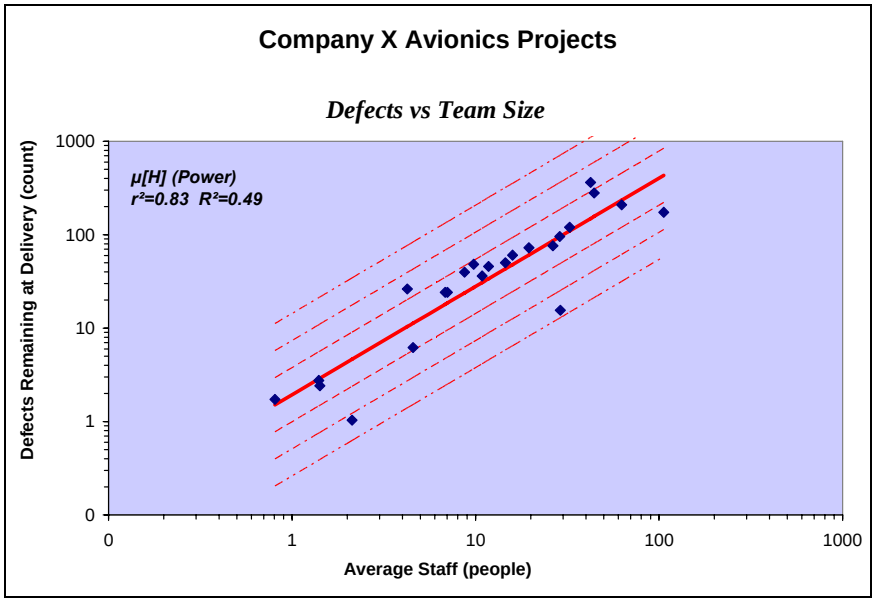

Figure 5. Defects Increase as Team Size Increases

***Projects seek a balance*—**Irrespective of size, duration and effort are reasonably correlated, which suggests some inherent equilibrium between the two (see Figure 6 below).

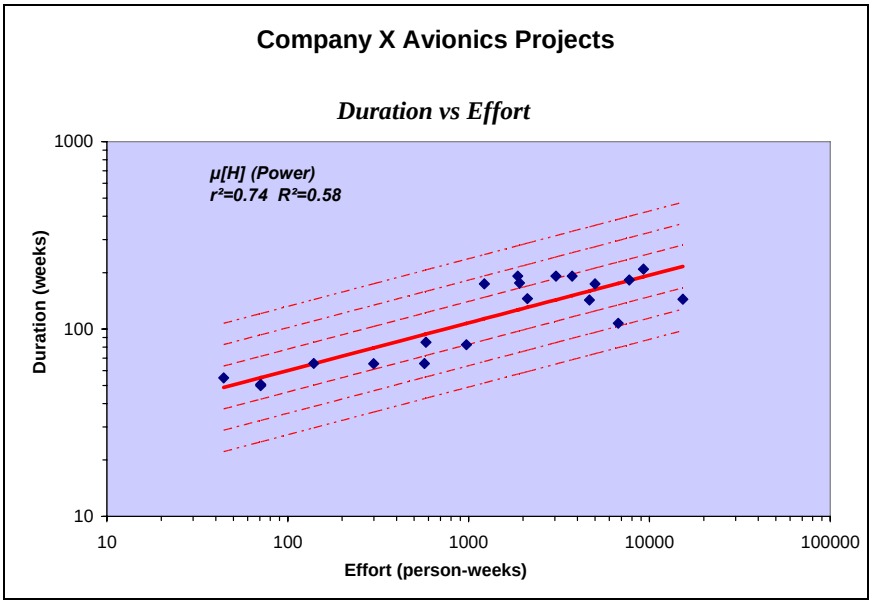

Figure 6. Equilibrium Between Duration and Effort

Fundamental Empirically-Verified Hypotheses

Software can be estimated as a multiplicative relationship between labor and time—The construction of software requires people to do work over some period of time. For a given project environment (people, process, and product):

- Adding effective software size increases the effort (implied by Figure 1) and/or duration (implied by Figure 2).
- Adding effort increases the potential effective software size (implied by Figure 1) and/or reduces the duration (implied by Figure 4 keeping in mind the inverse power relationship between productivity and average staff where the exponent on reciprocal average staff is less than 1).
- Adding duration increases the potential effective software size (implied by Figure 2) and/or reduces the effort (implied by Figure 4 keeping in mind the inverse power relationship between productivity and average staff where the exponent on reciprocal average staff is less than 1).

Defects can be estimated as a ratio relationship between labor and time—Defects in a software product are the unwanted byproduct of people attempting to do work over some period of time. For a given project (people, process, and product):

- Adding effort increases the number of defects (implied by Figure 3).
- Adding duration decreases the number of defects (implied by Figure 5).

Combining the two above-mentioned hypotheses for a given project environment (people, process, and product) implies the following intuitively-reasonable corollary truisms:

- Larger software products generally experience more defects than do smaller software products.
- Attempting to achieve a compressed schedule by adding people to a given project generally increases the number of defects experienced.
- Relaxing the schedule constraint of a given project offers the opportunity, within limits, to reduce cost (less effort) and improve product reliability (fewer defects).

3. DEFINING EFFORT, DURATION, SIZE, AND PRODUCTIVITY

Staffing, Duration, and Construction Duration

We first conceptually define staffing to be some function P of elapsed calendar time t that describes, for a particular instance of a software construction process, the application of people over time within the software construction time interval. We define this time interval in absolute terms as $[\mathrm{T}_{\text{start}}, \mathrm{T}_{\text{finish}}]$ where $\mathrm{T}_{\text{start}}$ and $\mathrm{T}_{\text{finish}}$ represent the start and finish dates of the construction process. In the interest of generalization, we prefer to use a $\mathrm{T}_{\text{start}}$-relative frame to describe this interval; therefore, $\mathrm{T}_{\text{start}}$ relative to $\mathrm{T}_{\text{start}}$ is $\mathrm{T}_{\text{start}} - \mathrm{T}_{\text{start}} = 0$ and $\mathrm{T}_{\text{finish}}$ relative to $\mathrm{T}_{\text{start}}$ is $\mathrm{T}_{\text{finish}} - \mathrm{T}_{\text{start}}$, the value of which we will represent as t_C. The resulting $\mathrm{T}_{\text{start}}$-relative construction interval is $[0, t_C]$. Note that the value of t_C represents not only the $\mathrm{T}_{\text{start}}$-relative point in time where construction finishes, it also represents the duration (elapsed calendar time) of the construction interval.

Effort

With our conceptual definition of a staffing function P, we now define the concept of effort to be some function E of elapsed calendar time t that describes, for a particular project, the accumulated result of people doing work over elapsed time t.

$$\mathrm{E} \equiv \int \mathrm{P}\, dt \tag{9}$$

Using Equation (9) as our definition of an effort function with respect to its associated staffing function we can now define an instantaneous staffing function with respect to its associated effort function by solving Equation (9) for P.

$$\begin{aligned} d\,\mathrm{E} &= d \int \mathrm{P}\, dt \\ \mathrm{P} &= \frac{d}{dt}\mathrm{E} \end{aligned} \tag{10}$$

Construction Effort

We have already defined $[0, t_C]$ to be the T_{start}-relative construction time interval where t_C represents construction duration. We now define construction effort E_C to be the change in effort within this construction time interval.

$$E_C \equiv E(t_C) - E(0) \tag{11}$$

Effective Software Size

The software construction process transforms one abstraction (the desire or the requirements) to another abstraction (the software product). Each and every abstraction, be it expressed in a natural language, a programming language, or even as graphic constructs; consists of expression primitives that we refer to as ***expression units*** (EUs). We choose to define the notion of effective software size S_e of a particular abstraction to be the number (count) of EUs in the abstraction that are considered to be directly related to the work associated with some software construction activity; this work including that associated with developing new software plus that associated with selecting, understanding, incorporating, changing, and/or verifying legacy software.

Average Construction Productivity

We can now define average software construction productivity ϖ to be the average amount of effective software size S_e that is transformed per unit of effort. In other words, average productivity is defined as the average effective software size developed per unit of effort; this being done over the interval of construction $[0, t_C]$.

$$\varpi \equiv \frac{S_e}{E_C} \tag{12}$$

4. EFFORT-DURATION TRADEOFF RELATIONSHIP

Software Construction Process Law

Software is made by people doing work over some period of time; the result being neither free, instant, nor perfect. We have already shown that effort E_C and duration t_C increase monotonically (and in most cases non-linearly) as functions of increasing effective software size (see Figure 1 and Figure 2). Our first empirically-verified hypothesis states that ***software can be estimated as a multiplicative relationship between effort and duration***. We therefore propose the following generalized relationship:

$$\begin{aligned} f_E(E_C)\, f_t(t_C) &\propto f_S(S_e) \text{ or} \\ f_E(E_C)\, f_t(t_C) &= b f_S(S_e) \end{aligned} \tag{13}$$

where b represents the constant of proportionality.

Performing ordinary (linear) least squares (OLS) regression in log-log space on data from past projects indicates that both effort and duration are reasonably correlated with effective software size (see Figure 1 and Figure 2). These correlations can be generally and reasonably modeled by power functions described as

$$\begin{aligned} f_{\mathrm{E}}(x) &\equiv x^{a_E} \\ f_{\mathrm{t}}(x) &\equiv x^{a_t} \\ f_{\mathrm{S}}(x) &\equiv x^{a_S} \end{aligned} \tag{14}$$

Substituting Equations (14) into Equation (13) yields

$$E_C{}^{a_E} t_C{}^{a_t} = b\left(S_e\right)^{a_S} \tag{15}$$

We then scale the resulting optimized exponent values a_E and a_t to force the exponent on size to unity by letting

$$\begin{aligned} a_E &\equiv \alpha_E a_S \\ a_t &\equiv \alpha_t a_S \end{aligned} \tag{16}$$

Note the distinction between the English letters "*a*" and the Greek letters α (alpha). Substituting Equations (16) into Equation (15) yields

$$\begin{aligned} E_C{}^{\alpha_E a_S} t_C{}^{\alpha_t a_S} &= b\left(S_e\right)^{a_S} \\ E_C{}^{\alpha_E} t_C{}^{\alpha_t} &= b^{\left(\frac{1}{a_S}\right)} S_e \end{aligned} \tag{17}$$

We next introduce the concept of data sample mean efficiency $\bar{\eta}$ (Greek lower-case eta bar) and choose to define it as being proportional to the reciprocal of the coefficient on effective software size

$$\begin{aligned} \frac{1}{b^{\left(\frac{1}{a_s}\right)}} &\propto \bar{\eta} \\ b^{\left(\frac{1}{a_s}\right)} &= \frac{1}{k_\eta \bar{\eta}} \end{aligned} \tag{18}$$

where k_η is a proportionality constant that resolves the system of units being used; its value being unity when effort is measured in person-weeks and duration is measured in calendar weeks.

Substituting Equation (18) into Equation (17) yields

$$E_C^{\alpha_E} t_C^{\alpha_t} = \left(\frac{1}{k_\eta \bar{\eta}}\right) S_e$$
$$E_C^{\alpha_E} t_C^{\alpha_t} = \frac{S_e}{k_\eta \bar{\eta}} \quad (19)$$

Values for α_E, α_t, and $\frac{1}{k_\eta \bar{\eta}}$ in Equation (19) for a specific historical data set can be empirically determined by performing two OLS regressions in log-log space on:

- Actual Effort versus Actual Effective software size (an example of which is shown in Figure 1)
- Actual Duration versus Actual Effort (an example of which is shown in Figure 6)

The resulting two power functions are

$$E_C = b_1 S_e^{a_1} \quad (20)$$

and

$$t_C = b_2 E_C^{a_2} \quad (21)$$

Multiplicatively combining Equation (20) and Equation (21) yields

$$E_C t_C = b_1 S_e^{a_1} b_2 E_C^{a_2}$$
$$E_C^{\left(\frac{1-a_2}{a_1}\right)} t_C^{\left(\frac{1}{a_1}\right)} = \frac{S_e}{\left(\frac{1}{b_1 b_2}\right)^{\left(\frac{1}{a_1}\right)}} \quad (22)$$

Finally, instantiating the general form Equation (19) with the regression-derived Equation (22) implies the following assignments:

$$\alpha_E = \frac{1-a_2}{a_1} \quad (23)$$

$$\alpha_t = \frac{1}{a_1} \quad (24)$$

$$\bar{\eta} = \left(\frac{1}{b_1 b_2}\right)^{\left(\frac{1}{a_1}\right)} \quad (25)$$

Figure 7 tests the correlation implied by Equation (22) using the example data of Figure 1 and Figure 6. Note the relatively high resulting r^2 and R^2 values which are indications of a reasonably good estimating relationship.

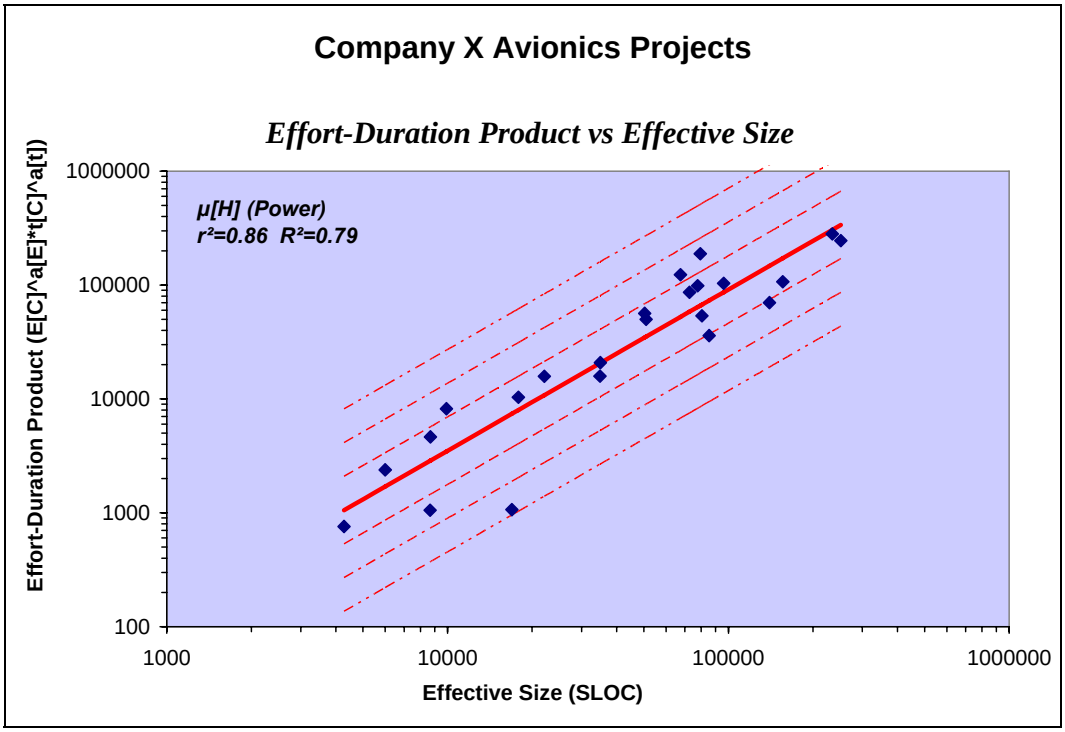

Figure 7. Effort-Duration Product versus Size Correlation

Specific Efficiency

We now introduce the notion of a project's specific efficiency η (Greek lower-case eta). Each instance of the software construction process has a unique value for specific efficiency within the context of a particular instantiation of Equation (19). Practically speaking, this means a value for specific efficiency of a project relates that project to the other projects in the particular historical data set for which Equation (19) has been instantiated and from which this value for specific efficiency has been determined. This project-specific value for specific efficiency can be determined in one of two ways:

- ***Empirically***—selected to be consistent with specific efficiency values calculated from the final actuals of relevant previously-completed projects (often referred to as calibration) or
- ***Parametrically***—calculated as the result of biasing mean efficiency $\bar{\eta}$ up or down as a function of weighted and normalized environmental attributes (sometimes referred to as parameters, effort drivers, or cost drivers), each having been shown to significantly influence a project's specific efficiency.

$$\eta = \bar{\eta} \prod_{i=1}^{n} Parameter_i \tag{26}$$

Effort-Duration Equation in Terms of the Software Product

If we instantiate Equation (19) (either empirically or parametrically) to fit the circumstances of a particular project, we get

$$E_C^{\alpha_E} t_C^{\alpha_t} = \frac{S_e}{k_\eta \eta} \tag{27}$$

Equation (27) is the ***fundamental software productivity equation***. It describes the tradeoff relationship between total construction effort and total construction duration as a function of the project's expected effective software size and the project's expected specific efficiency.

Figure 8 illustrates an example of the software productivity equation instantiated for a particular project's tradeoff relationship exponents α_E and α_t and for its effective software size S_e and its specific efficiency η.

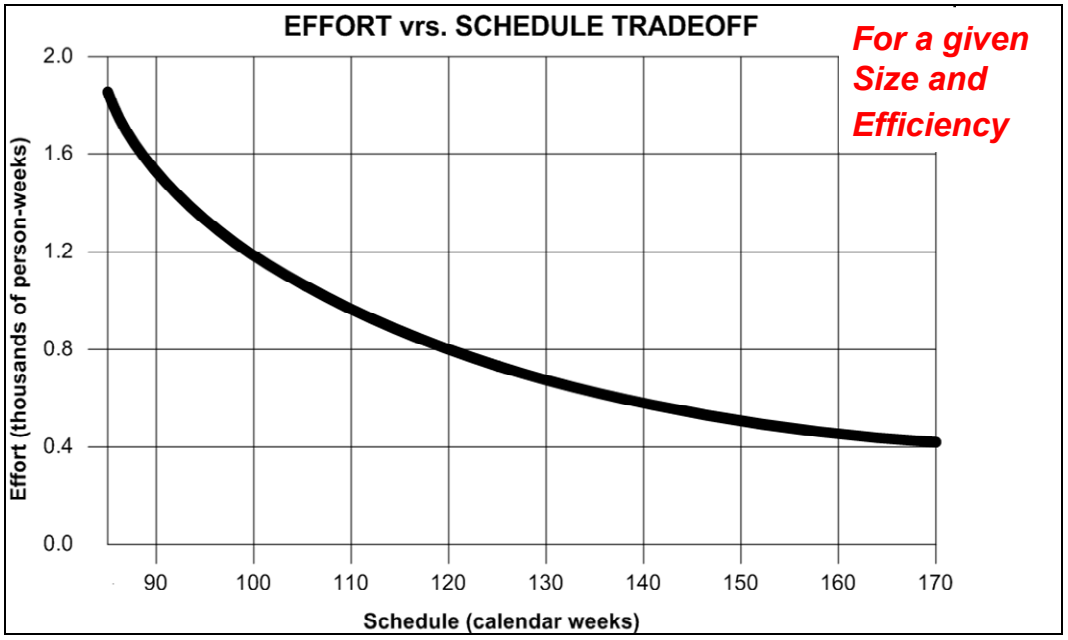

Figure 8. Example Project Tradeoff Curve (Software Productivity Law)

Solving for Construction Effort

Solving Equation (27) for construction effort we get

$$E_C = \left(\frac{S_e}{k_\eta \eta t_C^{\alpha_t}} \right)^{\left(\frac{1}{\alpha_E}\right)} \tag{28}$$

Solving for Construction Duration

Solving Equation (27) for construction duration we get

$$t_C = \left(\frac{S_e}{k_\eta \eta E_C^{\alpha_E}} \right)^{\left(\frac{1}{\alpha_t}\right)} \tag{29}$$

Solving for Efficiency (Calibration Form)

Solving Equation (27) for specific efficiency we get

$$\eta = \frac{S_e}{k_\eta E_C^{\alpha_E} t_C^{\alpha_t}} \tag{30}$$

Solving for Effective Software Size

Solving Equation (27) for effective software size we get

$$S_e = k_\eta \eta E_C^{\alpha_E} t_C^{\alpha_t} \tag{31}$$

5. DESCRIBING A PARTICULAR EFFORT-DURATION SOLUTION

Management Stress

The notion of management stress was suggested by Jensen [5] and described as the inherent equilibrium between effort and duration for the software construction process, this equilibrium being independent of effective software size and specific efficiency and being constrained by the earlier-described Rayleigh-shape staffing assumption (see footnote 8).

We choose to redefine this notion of management stress by eliminating the Rayleigh-shape staffing assumption constraint and by more-generally postulating that construction effort E_C is proportional to some function $\boldsymbol{f}$ of construction duration t_C. In other words, for the set of all projects, ignoring the variety of effective software sizes and of specific efficiencies, as the construction duration increases, the construction effort increases and *vice versa*. Stated mathematically

$$\begin{aligned} E_C &\propto \boldsymbol{f}\left(t_C\right) \text{ or} \\ E_C &= b\boldsymbol{f}\left(t_C\right) \end{aligned} \tag{32}$$

where b represents the constant of proportionality.

Performing OLS regression in log-log space ($\log\left(E_C\right)$ versus $\log\left(t_C\right)$) (inverting the regression shown earlier in Figure 6) on various past project data sets (neither stratified by size nor efficiency) indicates that construction effort increases monotonically (and in most cases non-linearly) as a function of increasing construction duration (see Figure 9 below). This correlation

can be generally and reasonably modeled by a power function of the form $f = x^a$. Substituting t_C for x yields $f(t_C) = t_C{}^a$. Finally, substituting this into Equation (32) and renaming a to γ (Greek lower-case gamma) yields

$$E_C = bt_C{}^{\gamma} \tag{33}$$

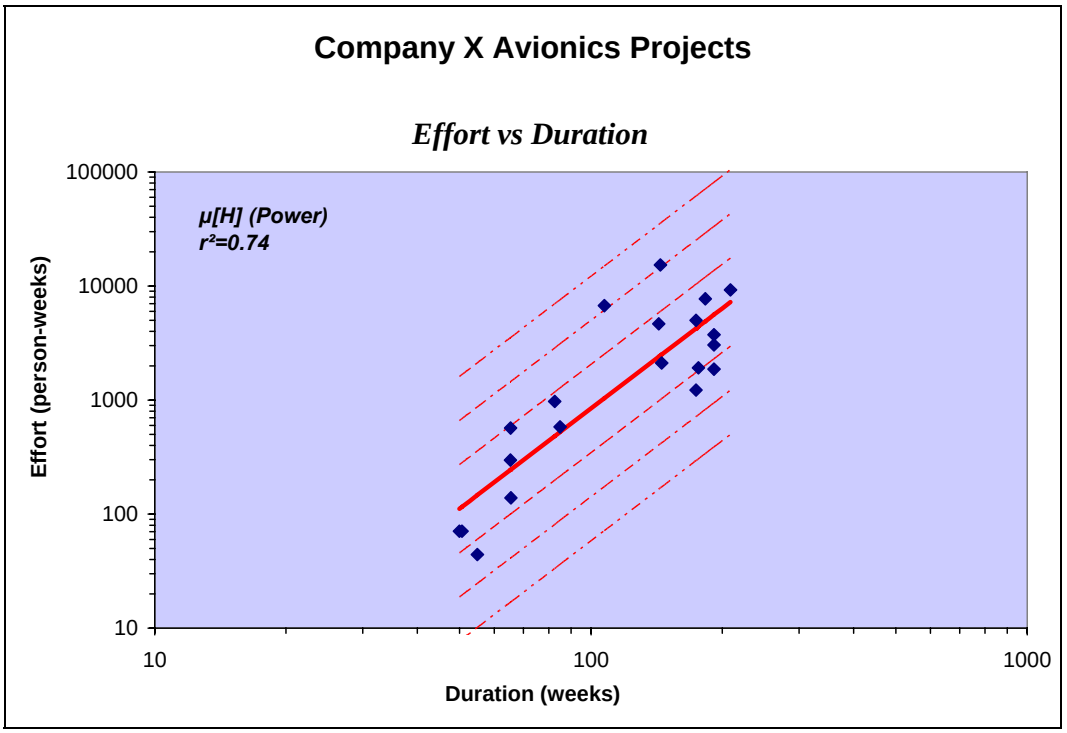

Figure 9. Effort Increases as Duration Increases

Note that Type 2 models such as COCOMO [1] imply $\gamma \equiv 1/a_2$. Note also that Jensen [5] and Putnam [10] both assume $\gamma \equiv 3$ (a constant).

Practically speaking, Equation (33) implies that as the construction duration is increased, the resultant construction effort increases, this increase characterized by the parameters γ and b. From a practical standpoint, γ represents the ***economy*** or diseconomy associated with higher construction durations and b is directly related to the data sample mean management stress $\bar{M}$. This proportionality resolves as follows

$$\begin{aligned} b &\propto \bar{M} \\ b &= k_M \bar{M} \end{aligned} \tag{34}$$

where k_M is a proportionality constant that resolves the system of units being used; its value being unity when effort is measured in person-weeks and duration is measured in calendar weeks.

Specific Management Stress

We now introduce the notion of a project's specific management stress M. Given a data set with a sample mean management stress of $\bar{M}$ and an economy of γ, each instance of the software construction process has unique specific management stress M, this value being tied to that particular data set and its value for γ. Specific management stress is limited by environmental (personnel, process, and product) characteristics; this limit can be determined empirically or parametrically (similar to the previously-described process for determining specific efficiency). A parametric determination takes the form of

$$M = \bar{M} \prod_{i=1}^{n} Parameter_i \tag{35}$$

Size-Independent Effort-Duration Equation

If we perform the above-described regression analysis on the final actuals from a reasonably relevant set of completed software construction instances, determine the resultant values for parameters γ and b, use Equation (34) to determine $\bar{M}$, and then parametrically instantiate Equation (33) with Equation (35), the result is:

$$\begin{aligned} E_C &= b t_C^{\gamma} \\ E_C &= k_M \bar{M} \prod_{i=1}^{n} (Parameter_i) t_C^{\gamma} \\ E_C &= k_M M t_C^{\gamma} \text{ or} \\ t_C &= \left(\frac{E_C}{k_M M} \right)^{\left(\frac{1}{\gamma}\right)} \text{ or} \\ M &= \frac{E_C}{k_M t_C^{\gamma}} \end{aligned} \tag{36}$$

Equation (36) is the ***fundamental software management stress equation***. It describes the relationship between total construction effort and total construction duration for a given project. It will be used in the next two sections to isolate particular effort-duration solutions for given effective software size and specific efficiency. It will also be used later in the paper as the basis for determining the feasible limits of the software productivity equation (minimum duration solution and minimum effort solution).

Solving for Construction Duration

Substituting the solved-for-effort form of Equation (36) into Equation (27) we get construction duration, this being a function of the specific management stress, the effective software size, and the specific efficiency.

$$\left(k_M M t_C{}^{\gamma}\right)^{\alpha_E} t_C{}^{\alpha_t} = \frac{S_e}{k_\eta \eta}$$

$$t_C = \left(\frac{1}{k_M M}\right)^{\left(\frac{\alpha_E}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{1}{\gamma\alpha_E+\alpha_t}\right)} \tag{37}$$

Solving for the Construction Effort Associated with a Particular Construction Duration

Substituting the solved-for-time form of Equation (36) into Equation (37) we get the construction effort associated with a given construction duration as a function of the effective management stress, the effective software size, and the specific efficiency.

$$\left(\frac{E_C}{k_M M}\right)^{\left(\frac{1}{\gamma}\right)} = \left(\frac{1}{k_M M}\right)^{\left(\frac{\alpha_E}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{1}{\gamma\alpha_E+\alpha_t}\right)}$$

$$E_C = \left(k_M M\right)^{\left(\frac{\alpha_t}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{\gamma}{\gamma\alpha_E+\alpha_t}\right)} \tag{38}$$

6. MINIMUM DURATION

Brooks' Law

Adding manpower to a late software project makes it later. [4] Each and every instance of the software construction process, by its nature (divisibility or potential for concurrency), can effectively handle only so much management stress (only so many people); therefore, there exists, for each and every instance of the software construction process, some minimum achievable construction duration. Software construction, like the aging of a fine wine, takes time and cannot be rushed beyond a ***certain point***.[11] Jensen [5], Putnam [10], and others have analyzed historical project data and concluded that this ***certain point*** can be defined in terms of a project's maximum achievable specific management stress given its degree of technical difficulty.

Maximum-Achievable Specific Management Stress

For each instance of the software construction process (a given project with given effective software size and given specific efficiency), there exists a maximum-achievable specific management stress value $M_{\max}$. In other words, each project, by its nature (divisibility or potential for concurrency), can effectively handle only so many additional people at a given time.

[11] Analogy frequently used by Dr. Randy Jensen in numerous presentations on this topic.

Qualitatively, maximum-achievable specific management stress is proportional to some function of a project's potential for parallelism; specifically, the speed with which the problem can be decomposed (divided and assigned to additional staff, the resulting work being performed in parallel). Keeping in mind the typical inverse relationship between maximum specific management stress and technical difficulty, this implies that more difficult projects (lower possible specific management stress values) tend toward serial task dependencies while less difficult projects (higher possible specific management stress values) allow for more parallel task dependencies.

Mathematically, the notion of a maximum-achievable value for specific management stress $M_{\max}$ can be expressed as

$$M_{\max} \geq M \tag{39}$$

Since this notion of a *maximum-achievable* value is somewhat theoretical, we suggest a conservatively-reasonable value for $M_{\max}$ in estimation situations is the y-intercept of the $+1\sigma$ trend line (e.g., the first dashed line above the solid line in Figure 9) of the OLS regression applied to $\log(E_C)$ versus $\log(t_C)$ from the relevant historical data set.

Substituting Equation (36) into Equation (39) yields

$$M_{\max} \geq \frac{E_C}{k_M t_C^{\gamma}} \text{ or}$$
$$E_C \leq M_{\max} k_M t_C^{\gamma} \tag{40}$$

Figure 10 shows the region excluded by Equation (40) in red given a value for maximum specific management stress $M_{\max}$. Note that the curve described by the margin between the red region and the white region is the minimum duration limiting function (Equation (40) as an equality).

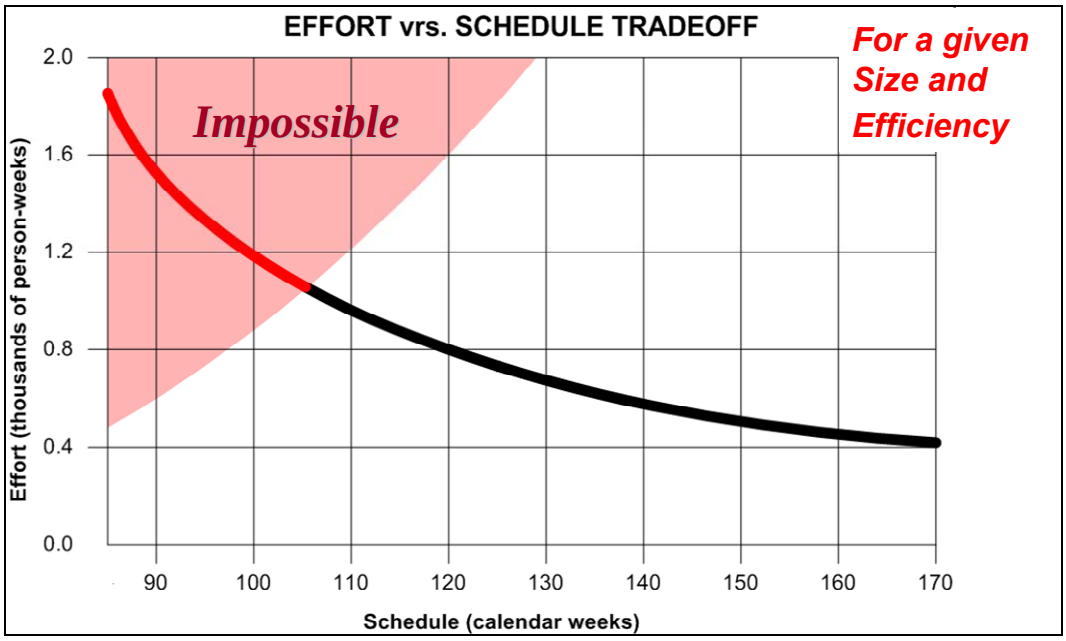

Figure 10. Minimum Duration Limit

Since maximum-achievable specific management stress $M_{\max}$ bounds the maximum amount of parallelism that can be incorporated into project scheduling, Equation (40) implies the existence of a minimum-achievable duration $t_{C\min}$ with associated effort $E_{t_{C\min}}$.

Solving for Minimum Construction Duration

Instantiating Equation (37) with $t_{C\min}$ and $M_{\max}$ we get the minimum construction duration, this being a function of the maximum-achievable specific management stress, the effective software size, and the specific efficiency.

$$t_{C\min} = \left(\frac{1}{k_M M_{\max}}\right)^{\left(\frac{\alpha_E}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{1}{\gamma\alpha_E+\alpha_t}\right)} \tag{41}$$

Solving for Effort Associated with Minimum Construction Duration

Instantiating Equation (38) with $E_{t_{C\min}}$ and $M_{\max}$ we get the effort associated with the minimum construction duration, this being a function of the maximum-achievable specific management stress, the effective software size, and the specific efficiency.

$$E_{t_{C\min}} = \left(k_M M_{\max}\right)^{\left(\frac{\alpha_t}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{\gamma}{\gamma\alpha_E+\alpha_t}\right)} \tag{42}$$

Figure 11 shows the minimum construction duration and its associated effort as one of the solutions on the effort versus schedule (duration) tradeoff curve. Note that the minimum duration solution is defined as the intersection of the fundamental software productivity equation (Equation (28)) and the Brooks' Law or minimum duration limiting function (Equation (40) as an equality)).

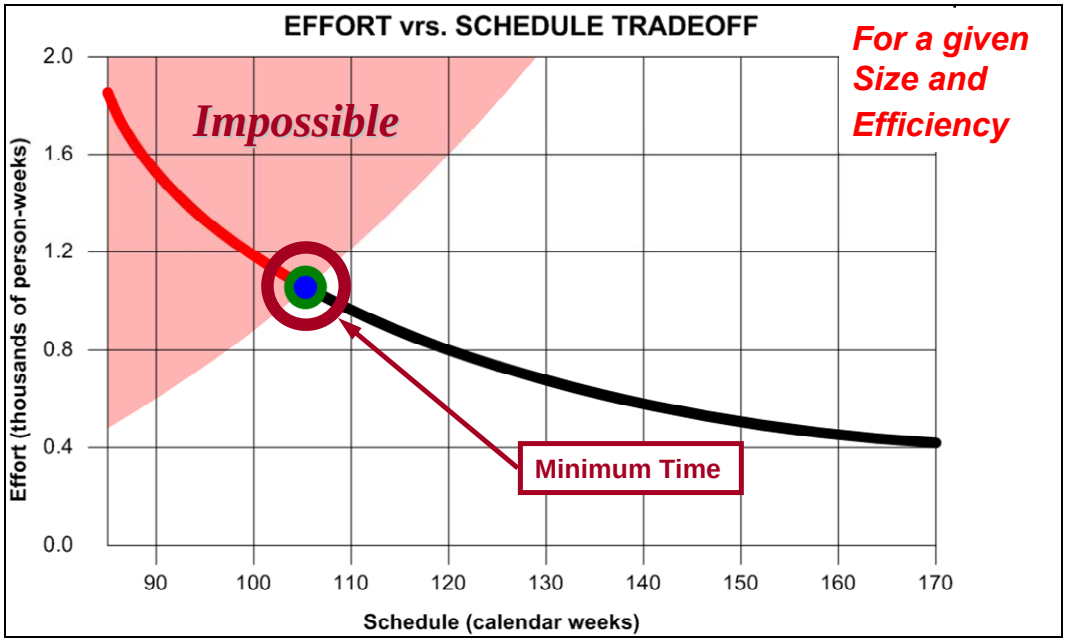

Figure 11. Minimum Duration Solution

7. MINIMUM EFFORT

Parkinson's Law

Work expands so as to fill the time available for its completion. [9] Theoretically, a specific instance of a software construction process is not limited by some maximum construction duration. Rare is the software engineer who complains about having too much time to develop software. However, we submit that there exists, for each and every project, some duration that yields maximum productivity; i.e., some duration that represents the most efficient combination of project decomposition and corresponding use of labor.

Minimum-Practical Specific Management Stress

For each instance of the software construction process, we submit that maximum productivity occurs at some point of minimum-practical specific management stress. This point of minimum practical specific management stress $M_{\min}$ defines the optimum use of people over time, and represents a practical limit to the benefit of schedule relaxation.

Mathematically, the notion of a minimum value for specific management stress $M_{\min}$ can be expressed as

$$M_{\min} \le M \tag{43}$$

Since this notion of a *minimum-practical* value is somewhat theoretical, we suggest a conservatively-reasonable value for $M_{\min}$ in estimation situations is the y-intercept of the -1σ trend line (e.g., the first dashed line below the solid line in Figure 9) of the OLS regression applied to $\log(E_C)$ versus $\log(t_C)$ from the relevant historical data set.

Substituting Equation (36) into Equation (43) yields

$$M_{\min} \le \frac{E_C}{k_M t_C^{\gamma}} \text{ or}$$
$$E_C \ge M_{\min} k_M t_C^{\gamma} \tag{44}$$

Figure 12 shows the region excluded by Equation (44) in yellow given a value for minimum specific management stress $M_{\min}$. Note that the curve described by the margin between the yellow region and the white region is the minimum effort limiting function (Equation (44) as an equality).

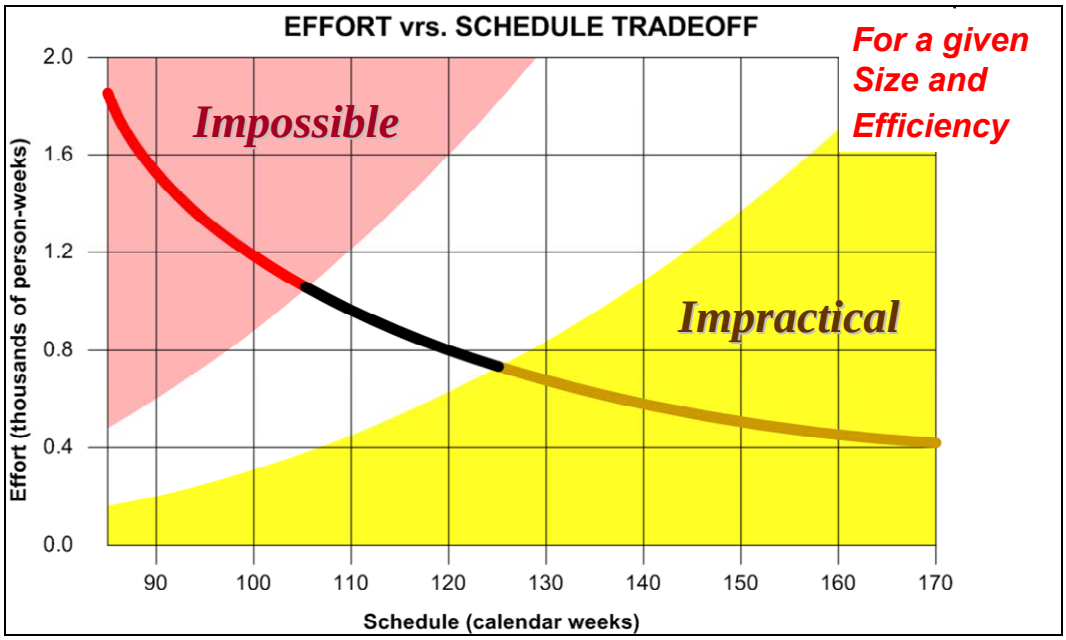

Figure 12. Minimum Effort Limit

Since minimum specific management stress $M_{\min}$ bounds the minimum effective amount of project parallelism that can be assumed before Parkinson's law overrides the cost benefit of sched-

ule relaxation, Equation (44) implies the existence of a minimum-achievable effort $E_{C\min}$ with associated duration $t_{E_{C\min}}$.

Solving for Minimum Construction Effort

Instantiating Equation (38) with $E_{C\min}$ and $M_{\min}$ we get the minimum construction effort, this being a function of the minimum specific management stress, the effective software size, and the specific efficiency.

$$E_{C\min} = \left(k_M M_{\min}\right)^{\left(\frac{\alpha_t}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{\gamma}{\gamma\alpha_E+\alpha_t}\right)} \tag{45}$$

Solving for Duration Associated with Minimum Construction Effort

Instantiating Equation (37) with $t_{E_{C\min}}$ and $M_{\min}$ we get the duration associated with the minimum construction effort, this being a function of the minimum specific management stress, the effective software size, and the specific efficiency.

$$t_{E_{C\min}} = \left(\frac{1}{k_M M_{\min}}\right)^{\left(\frac{\alpha_E}{\gamma\alpha_E+\alpha_t}\right)} \left(\frac{S_e}{k_\eta \eta}\right)^{\left(\frac{1}{\gamma\alpha_E+\alpha_t}\right)} \tag{46}$$

Figure 13 shows the minimum construction effort and its associated duration as one of the solutions on the effort versus schedule (duration) tradeoff curve. Note that the minimum effort solution is defined as the intersection of the fundamental software productivity equation (Equation (28)) and the Parkinson's Law or minimum effort limiting function (Equation (44) as an equality)).

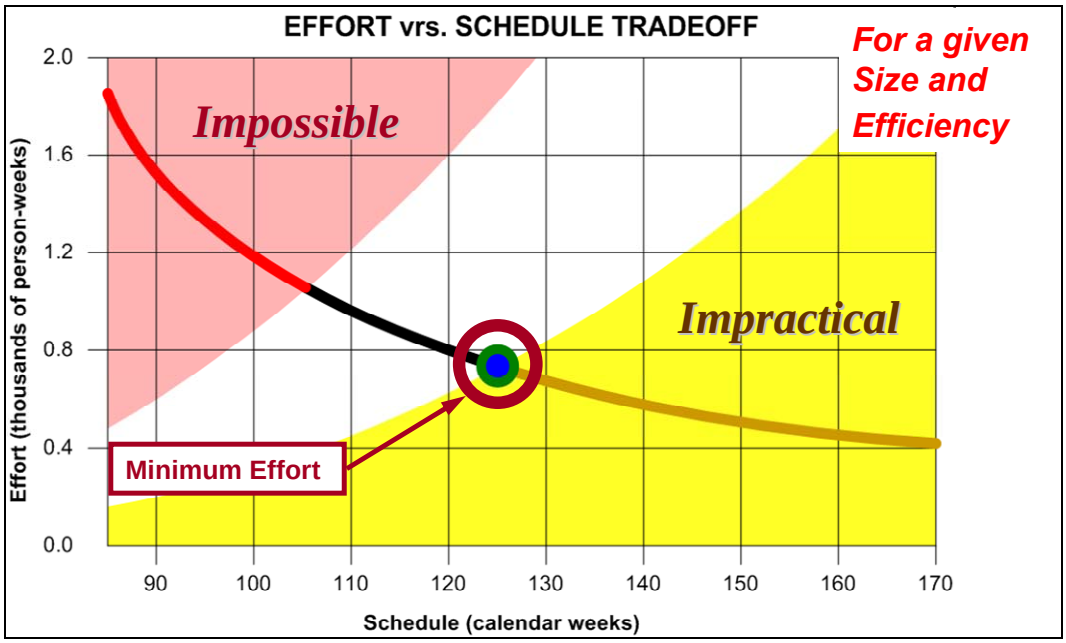

Figure 13. Minimum Effort Solution

8. DEFECTS

Taxonomy

In order to derive the defects estimating portion of our evolving model it is useful to establish definitions of the terminology associated with software reliability. For the most part, we choose to adopt Musa's [7] definitions with minor modification:

Failure—"…departure of the external results of system operation from user [desires]."[12]

Fault—"…defective, missing, or extra [expression unit] or set of related [expression units] that is the cause of one or more actual or potential failures."[13]

Error—"…incorrect or missing action by a person or persons that causes a fault in a program."[14]

Problem or Issue—The *documented* observation of some undesired aspect or aspects of a software product and/or its related documentation; typically contained in the form of an artifact; e.g., problem report, trouble report, issue report, etc. Problems, according to this definition, are very imprecise (such is the nature of problem reporting systems) and can contain, within their scope, various combinations of failures, faults, errors, and suggested enhancements. Duplicates and overlaps are not uncommon.

---

[12] [7] p. 208
[13] [7] p. 213
[14] [7] p. 215

Defect—The *documented* result of a problem review process that seeks to identify and isolate the fault(s) associated with a particular problem. This definition implies a review process with one of its objectives being to achieve a near one-to-one correspondence between defects and faults.

Unacceptable Defect—A defect, the removal of which, is considered a necessary condition for software product acceptance (e.g., certification, customer buy-off, general availability, contract satisfaction, etc.).

Defect Occurrence Function

We first conceptually define cumulative defect occurrence to be some function Φ (Greek upper-case phi) of elapsed calendar time t that describes, for a particular project, the accumulation of discovered defects over elapsed calendar time within the software construction time interval $[0, t_C]$.

Defect Occurrence Rate Function

With our conceptual definition of cumulative defect occurrence Φ, we now define the concept of defect occurrence rate (sometimes referred to as instantaneous defect occurrence) to be some function $\dot{\Phi}$ of elapsed calendar time t that describes, for a particular project, the rate that defects are being discovered at a particular point in time within the construction time interval $[0, t_C]$.

$$\dot{\Phi} \equiv \frac{d}{dt}\Phi \tag{47}$$

which implies

$$\Phi = \int \dot{\Phi}\, dt \tag{48}$$

Software Construction Process Law (Revisited)

Software is made by people doing work over some period of time; the result being neither free, instant, nor perfect. We have already concluded, from our fundamental observations, that defect count $\Phi_{[a,b]}$ (number of defects discovered during the T_{start}-relative time interval $[t_a, t_b]$) increases monotonically (and in most cases non-linearly) as a function of increasing effort. Practically speaking, this implies that the software construction process has two primary outputs: the deliverable product software and the defects that it contains; i.e., defects, like code, can be considered products (albeit undesirable) of the process. Our second empirically-verified hypothesis (introduced earlier in this paper) states that ***defects can be estimated as a ratio relationship between labor and time***. We therefore propose the following generalized relationship:

$$\frac{\boldsymbol{g}_{\mathrm{E}}\left(E_C\right)}{\boldsymbol{g}_{\mathrm{t}}\left(t_C\right)} \propto \boldsymbol{g}_{\Phi}\left(\Phi_{[a,b]}\right) \text{ or}$$
$$\frac{\boldsymbol{g}_{\mathrm{E}}\left(E_C\right)}{\boldsymbol{g}_{\mathrm{t}}\left(t_C\right)} = b\boldsymbol{g}_{\Phi}\left(\Phi_{[a,b]}\right) \tag{49}$$

where b represents the constant of proportionality.

Performing OLS regression in log-log space on data from past projects indicates that both effort and duration can change non-linearly as functions of total defects. This nonlinear behavior is best predicted (yields the highest correlation) by power functions described as

$$\begin{aligned} \boldsymbol{g}_{\mathrm{E}}\left(x\right) &\equiv x^{a_E} \\ \boldsymbol{g}_{\mathrm{t}}\left(x\right) &\equiv x^{a_t} \\ \boldsymbol{g}_{\Phi}\left(x\right) &\equiv x^{a_\Phi} \end{aligned} \tag{50}$$

Substituting Equations (50) into Equation (49) yields

$$E_C{}^{a_E} t_C{}^{-a_t} = b\Phi_{[a,b]}{}^{a_\Phi} \tag{51}$$

We then scale the exponents to force the exponent on total defects to unity by letting

$$\begin{aligned} a_E &\equiv \varphi_E a_\Phi \\ -a_t &\equiv \varphi_t a_\Phi \end{aligned} \tag{52}$$

Substituting Equations (52) into Equation (51) yields

$$\begin{aligned} E_C{}^{\varphi_E a_\Phi} t_C{}^{\varphi_t a_\Phi} &= b\Phi_{[a,b]}{}^{a_\Phi} \\ E_C{}^{\varphi_E} t_C{}^{\varphi_t} &= b^{\left(\frac{1}{a_\Phi}\right)}\Phi_{[a,b]} \end{aligned} \tag{53}$$

We next introduce the concept of data sample mean defect vulnerability $\overline{\delta}_{[a,b]}$ (Greek lower-case delta bar) and choose to define it as being proportional to the reciprocal of the coefficient on total defects

$$\begin{aligned} \frac{1}{b^{\left(\frac{1}{a_s}\right)}} &\propto \overline{\delta}_{[a,b]} \\ b^{\left(\frac{1}{a_s}\right)} &= \frac{1}{k_\delta \overline{\delta}_{[a,b]}} \end{aligned} \tag{54}$$

where k_δ is a proportionality constant that resolves the system of units being used; its value being unity when effort is measured in person-weeks and duration is measured in calendar weeks.

Substituting Equation (54) into Equation (53) yields

$$\begin{aligned} E_C^{\varphi_E} t_C^{\varphi_t} &= \left(\frac{1}{k_\delta \bar{\delta}_{[a,b]}} \right) \Phi_{[a,b]} \\ E_C^{\varphi_E} t_C^{\varphi_t} &= \frac{\Phi_{[a,b]}}{k_\delta \bar{\delta}_{[a,b]}} \end{aligned} \tag{55}$$

Values for φ_E, φ_t, and $\frac{1}{k_\delta \bar{\delta}_{[a,b]}}$ in Equation (55) for a specific historical data set can be empirically determined by performing two OLS regressions in log-log space on:

- Actual Effort versus Actual Defects (an example of which is shown in Figure 3)
- Actual Duration versus Actual Effort (an example of which is shown in Figure 6)

The resulting two power functions are

$$E_C = b_3 \Phi_C^{a_3} \tag{56}$$

and

$$t_C = b_2 E_C^{a_2} \tag{57}$$

Ratio combining Equation (56) and Equation (57) yields

$$\begin{aligned} \frac{E_C}{t_C} &= \frac{b_3 \Phi_C^{a_3}}{b_2 E_C^{a_2}} \\ E_C^{\left(\frac{a_2+1}{a_3}\right)} t_C^{-\left(\frac{1}{a_3}\right)} &= \frac{\Phi_C}{\left(\frac{b_2}{b_3}\right)^{\left(\frac{1}{a_3}\right)}} \end{aligned} \tag{58}$$

Finally, instantiating the general form Equation (55) with the regression-derived Equation (58) implies the following assignments:

$$\varphi_E = \frac{a_2 + 1}{a_3} \tag{59}$$

$$\varphi_t = -\left(\frac{1}{a_3}\right) \tag{60}$$

$$\bar{\delta} = \left(\frac{b_2}{b_3}\right)^{\left(\frac{1}{a_3}\right)} \tag{61}$$

Figure 14 tests the correlation implied by Equation (58) using the example data of Figure 3 and Figure 6. Note the relatively high resulting r^2 and R^2 values which are indications of a reasonably good estimating relationship.

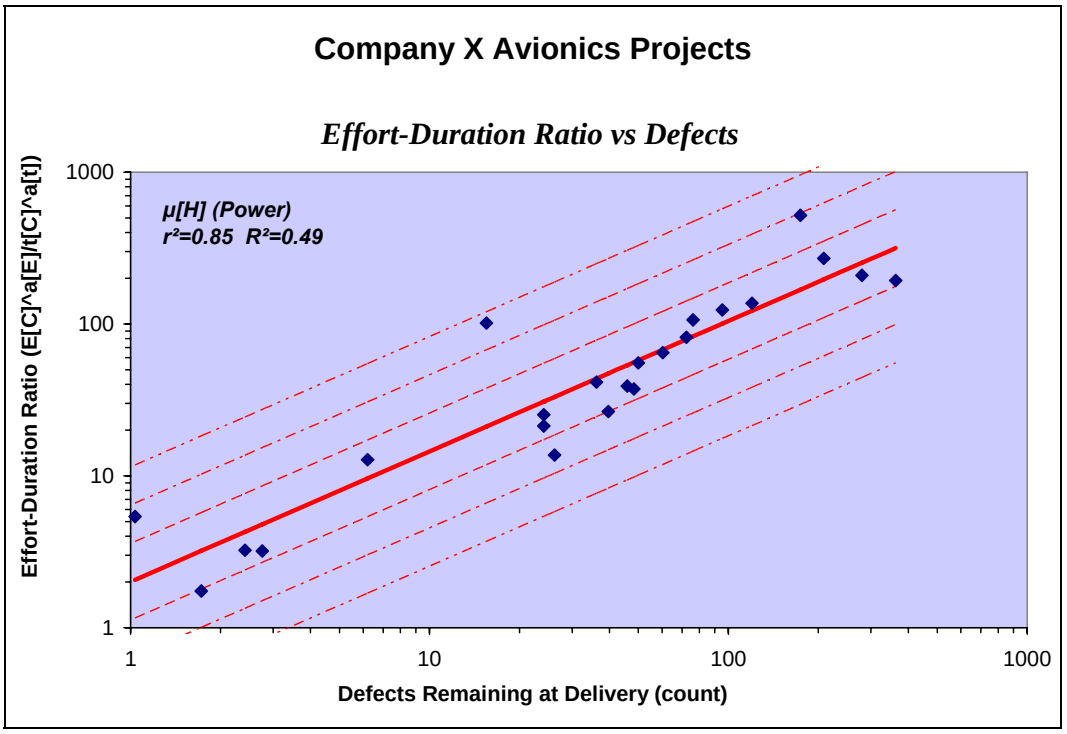

Figure 14. Effort-Duration Ratio versus Defects Correlation

Specific Defect Vulnerability

We now introduce the notion of a project's specific defect vulnerability $\delta_{[a,b]}$ (Greek lower-case delta). Each instance of the software construction process has a unique value for specific defect vulnerability that is driven by environmental (personnel, process, and product) characteristics. This project-specific value for specific defect vulnerability can be determined in one of two ways:

- ***Empirically***—selected to be consistent with specific defect vulnerability values calculated from the final actuals of relevant previously-completed projects (often referred to as calibration) or

- ***Parametrically***—calculated as the result of biasing mean defect vulnerability $\bar{\delta}_{[a,b]}$ up or down as a function of weighted and normalized environmental attributes (sometimes referred to as parameters, effort drivers, or cost drivers), each having been shown to significantly influence a project's specific defect vulnerability.

$$\delta_{[a,b]} = \bar{\delta}_{[a,b]} \prod_{i=1}^{n} Parameter_i \tag{62}$$

Effort-Duration Equation in Terms of Defects Produced

If we instantiate Equation (55) (either empirically or parametrically) to fit the circumstances of a particular project, we get

$$E_C{}^{\varphi_E} t_C{}^{\varphi_t} = \frac{\Phi_{[a,b]}}{k_\delta \delta_{[a,b]}} \tag{63}$$

Equation (63) is the ***fundamental software defect propensity equation***. It describes the relationship between total construction effort and total construction duration as a function of the project's expected defect count and the project's expected specific defect vulnerability.[15]

Solving for Construction Effort

$$E_C = \left(\frac{\Phi_{[a,b]}}{k_\delta \delta_{[a,b]} t_C{}^{-\varphi_t}} \right)^{\left(\frac{1}{\varphi_E}\right)} \tag{64}$$

Solving for Construction Duration

$$t_C = \left(\frac{\Phi_{[a,b]}}{k_\delta \delta_{[a,b]} E_C{}^{\varphi_E}} \right)^{\left(\frac{1}{\varphi_t}\right)} \tag{65}$$

Solving for Defects

$$\Phi_{[a,b]} = k_\delta \delta_{[a,b]} E_C{}^{\varphi_E} t_C{}^{\varphi_t} \tag{66}$$

Solving for Defect Vulnerability (Calibration Form)

$$\delta_{[a,b]} = \frac{\Phi_{[a,b]}}{k_\delta E_C{}^{\varphi_E} t_C{}^{\varphi_t}} \tag{67}$$

---

[15] The fairly general way in which defect count and defect vulnerability are defined allow for defect count to be specified within any range in the software construction process. Note that the scale of defect vulnerability is always associated with the particular defect count range. Note also that it is possible to use defect density at some particular point in the construction process in place of defect count as long as defect vulnerability is scaled accordingly.

Solving for Defects as a Function of Duration and Independent of Effort

Substituting the solved-for-effort form of Equation (36) into Equation (66) yields

$$\begin{aligned} \Phi_{[a,b]} &= k_\delta \delta_{[a,b]} \left(k_M M t_C{}^{\gamma} \right)^{\varphi_E} t_C{}^{\varphi_t} \\ \Phi_{[a,b]} &= k_\delta \delta_{[a,b]} \left(k_M M \right)^{\varphi_E} t_C{}^{(\gamma\varphi_E + \varphi_t)} \end{aligned} \tag{68}$$

Solving for Duration Independent of Effort

Solving Equation (68) for construction duration t_C yields

$$t_C = \left(\frac{\Phi_{[a,b]}}{k_\delta \delta_{[a,b]} \left(k_M M \right)^{\varphi_E}} \right)^{\left(\frac{1}{\gamma\varphi_E + \varphi_t} \right)} \tag{69}$$

Defects with Respect to Minimum Duration

Substituting Equation (64) into Equation (40) and solving for expected number of discovered defects during construction $\Phi_{[a,b]}$ yields

$$\begin{aligned} \left(\frac{\Phi_{[a,b]}}{k_\delta \delta_{[a,b]} t_C{}^{\varphi_t}} \right)^{\left(\frac{1}{\varphi_E} \right)} &\le M_{\max} k_M t_C{}^{\gamma} \\ \Phi_{[a,b]} &\le k_\delta \delta_{[a,b]} \left(M_{\max} k_M \right)^{\varphi_E} t_C{}^{(\gamma\varphi_E + \varphi_t)} \end{aligned} \tag{70}$$

Defects with Respect to Minimum Effort

Substituting Equation (64) into Equation (44) and solving for expected number of discovered defects during construction Φ_C yields

9. SUMMARY AND CONCLUSION

Purpose and Scope Revisited

This paper documents the basis, assumptions, and derivations behind a set of general software effort, duration, and defects estimating relationships that are based on the notion that software construction is the application of effort (labor) over some duration (period of elapsed calendar time) that produces a desired software product (size) and undesired byproducts (defects). The derivations, assumptions, and resulting model described by this paper establish relationships between size, efficiency, effort, duration, and defects and are collectively referred to as the ***Ross Software Estimating Framework***.

Areas for Further Study

The following are the author's observations and suggest possible opportunities for enhancing RSEF capabilities:

- The behavior of most currently-available software project estimating models (e.g., COCOMO 81 [1], COCOMO II [2], Jensen [5], and Putnam [10]) can be emulated by the RSEF equations given a corresponding instantiation of the RSEF variables α_E, α_t, γ, φ_E, φ_t, $M_{\min}$, M_{nom}, and $M_{\max}$. Instantiation values can be derived by algebraic manipulation of the particular model's estimating equation(s). We reasonably assume that the ability to emulate the behavior of multiple models plus the ability to estimate based on relevant historical data (both possible with the RESF) should provide increased overall estimate credibility.[16]
- The primary independent variables *effective software size*, *specific efficiency*, and *defect vulnerability* are uncertain until project completion. This suggests a stochastic dimension to the problem; i.e., effective software size, specific efficiency, and defect vulnerability should all be treated as random variables. The results of this treatment should be distributions of possible outcomes for effort, duration, cost, and discovered defects with associated confidence probabilities.
- Labor (the motive force), effective software size (the evolving product), and discovered defects are each dynamic; i.e., they can vary as the project progresses. This implies *estimates at completion* and *estimates to complete* are similarly dynamic. While the model described in this paper is adequate to estimate values for the total software construction process, it begs for a time-range differential calculus approach to provide in-process estimates once the project is under way.

[16] A recurring theme in recent International Society of Parametric Analysts conferences (particularly the "Renew Your Training" segments) has been the desire for multiple estimating model cross-checking in order to enhance estimate credibility and confidence.

REFERENCES

[1] Boehm, B., Software *Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

[2] Boehm, B. [et al.], *Software Cost Estimation with COCOMO II*, Prentice-Hall, Inc., Upper Saddle River, NJ, 2000.

[3] Book, S.A. and Young, P.H., "The Trouble with R^2," Journal of Parametrics, Vol. 25, No. 1 (Summer 2006).

[4] Brooks, F., *The Mythical Man-Month*, 20th Anniversary Edition, Addison-Wesley Publishing Company, Reading, MA, 1975, 1995.

[5] Jensen, R., "An Improved Macrolevel Software Development Resource Estimation Model," Software Engineering, Inc.

[6] Jensen, R., "Software Estimating Models: Three Viewpoints," CrossTalk, *The Journal of Defense Software Engineering*, STSC, Hill AFB, UT, May 1997.

[7] Musa, J., *Software Reliability Engineering: More Reliable Software Faster and Cheaper*, 2nd. Edition, AuthorHouse, Bloomington, IN, 2004.

[8] Norden, P., "Project Life Modeling: Background and Application of Life Cycle Curves," *Software Life Cycle Management Workshop*, Sponsored by USACSC, Airlie, VA, August 1977.

[9] Parkinson, C., *Parkinson's Law: The Pursuit of Progress*, John Murray, London, UK, 1958.

[10] Putnam, L. *Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers*, Computer Society Press, IEEE Computer Society, New York, NY, 1980.

[11] Ross, M., "Software Project Dynamics: A Productivity versus Staffing Approach," *Proc. ISPA 2006 Conference*, May 2006.

BIOGRAPHY

Mike Ross *has over 30 years of experience in software engineering as a developer, manager, process champion, consultant, instructor, and award-winning international speaker. Mr. Ross is currently the President and CEO of r2Estimating, LLC. Mr. Ross's previous experience includes three years as Chief Engineer of Galorath Inc. (makers of the SEER suite of estimation tools), seven years with Quantitative Software Management, Inc. (makers of the SLIM suite of software estimating tools) where he was Vice President of Education Services, and 17 years with Honeywell Air Transport Systems (formerly Sperry Flight Systems) and 2 years with Tracor Aerospace where he developed or managed the development of embedded software for avionics systems installed various commercial airplanes and for expendable countermeasures systems installed in various military aircraft. He also co-founded Honeywell Air Transport Systems' SEPG, served as its focal for software project management process improvement, and served as a Honeywell corporate SEI CMM assessor. Mr. Ross did his undergraduate work at the United States Air Force Academy and Arizona State University, receiving a Bachelor of Science in Computer Engineering.*