



An empirical study of ICASE learning curves and probability bounds for software development effort

Parag C. Pendharkar *, Girish H. Subramanian ¹

School of Business Administration, Penn State Harrisburg, 777 West Harrisburg Pike, Middletown, PA 17078, United States

Received 15 October 2004; accepted 15 July 2005

Abstract

In this paper, we investigate the existence of learning curves in software development. Under the assumption of independent and identical distribution (iid) of programmer's experience and identical effort-experience learning curve relationship for different programmers, we illustrate the existence of an exponentially decreasing learning curve relationship between a programmer's effort and his/her ICASE tool experience, and show that the effort-experience relationship is inelastic when a programmer's ICASE tool experience is low. We analyze the impact of our assumptions on actual software development effort, and propose a tight probability upper bound and a central-limit theorem based probability estimator for estimating the approximate probability that the software development effort will be less than or equal to a certain number. Examples to illustrate the use of the probability estimator are also provided.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Software engineering; Learning curve theory; Probability bounds; Chernoff bound; Neural networks; Central limit theorem; Moment generating functions

1. Introduction

Several researchers have characterized software development as an economic production process; many researchers have used non-parametric production function estimation, and forecasting tools to determine the relationships between inputs and outputs of the software development process [3,4,19]. The learning curve theory is a theory in the produc-

tion management literature which states that as production workers gain experience in the production tools their productivity increases. There is some anecdotal evidence in software engineering literature that suggests the existence of learning curves in software development and maintenance. For example, Banker and Slaughter [1] insinuating the existence of learning curves in software maintenance write, "...efficiency gains in maintenance occur where requests for a particular system are batched, so that the maintenance team can spread learning curve effects from becoming familiar with that system over several requests." The existence of learning curves in using ICASE tool was conjectured in Chew et al. [7] and Kemerer [13]. However, no study in the literature

* Corresponding author. Tel.: +717 948 6028; fax: +717 948 6456.

E-mail addresses: pxp19@psu.edu (P.C. Pendharkar), ghs2@psu.edu (G.H. Subramanian).

¹ Tel.: +1 717 948 6150; fax: +1 717 948 6456.

has empirically validated this generally accepted conjecture.

The empirical validation of the existence of learning curves in software engineering literature is a daunting task because many software companies do not collect the data necessary for testing the learning curve relationships [13]. Additionally, collection of necessary data is very difficult because of the dynamic nature of software development, where many different people with different level of experience and skills work on the development of a software product [16,13]. While it is difficult to establish the learning curve relationships between certain inputs and outputs in software production processes, the knowledge of their existence may help project managers/researchers improve their software development cost and effort estimates.

In the current study, we examine the relationship between a programmer experience in an integrated computer-aided software engineering (ICASE) tool and the software development effort. Under the assumption of independent and identical distribution (iid) programmer's experience and identical effort–experience learning curve relationship for different programmers, we hypothesize that the software development effort will reduce exponentially as programmers gain more experience in the usage of ICASE tools. We show that the reduction in development effort may be different for different ICASE tools, and the effort-experience relationship is inelastic when programmer's ICASE tool experience is low.

The rest of the paper is organized as follows. In Section 2, we present an introduction of the learning curve theory. In Section 3, we present our hypothesis and describe the data in Section 4. In Section 4, we test our hypothesis using an empirical data set. In Section 5, we analyze the impact our assumptions on actual software development effort, propose a tight probability upper bound, and a probability estimator for estimating the approximate probability for software development effort will be less than or equal to a certain number. In Section 6, we conclude our paper with summary and discussion.

2. An introduction to the learning curve theory

The learning curve theory was first introduced in aircraft industry, where it was found that as workers gained experience in the production of airplanes the time required to manufacture the airplanes decreased [10]. A learning curve expresses the decreasing time it

takes to perform a repetitive task as one continues from one cycle to another cycle of the task [9]. The original Wright learning curve (WLC) is represented as follows:

$$p(s) = \omega s^{-\gamma},$$

where ω is the time to manufacture the first unit, γ is the learning factor, s is the variable representing the number of units, and $p(s)$ is the time to produce the s th unit [25].

According to Nicks [18], every person learns at a different rate and the slope of learning curve may be different from one person to another. Nicks [18] argues that as quantity of product doubles, costs are reduced at some steady rate called the “learning rate.” A few researchers investigated the factors that impact the learning rate. According to Conway and Schultz [8], learning rate depends on two factors, pre-production and actual production. The pre-production factor includes tolling, equipment and tool selection, product design, work methods, and shop organization. The actual production factor includes work methods, design changes, quality improvements, planning, scheduling and control activities, and worker incentives. Finch and Luebbe [10] investigated the impact of learning rate and its variability on the project length. Their results indicate that project lengths are sensitive to learning rate. Thus, differences in learning rate may have an impact on the overall cost of a project.

Kevin Self [23] reports that there are two types of learning curves. These two types of learning curves are exponential and S-shaped. Both of these learning curves illustrate that the amount of knowledge gained starts off slowly, increases at an accelerating pace and then stabilizes. Both curves approach asymptotically to a “total understanding” point beyond which no-significant improvements in knowledge are materialized.

3. Literature review and hypothesis

There are four essential aspects of software development – tools, methodology, people, and management [11,22]. Research in [22] examined factors that impact software development productivity for projects using ICASE tools. Tools, methods, and personnel experience were indicated as variables that influence software development effort in the systems development effort model [26].

The impact of programmer ICASE tool experience on effort is known for quite some time now

[26,22]. For example, Wrigley and Dexter [26] mention that software engineers with higher experience are typically assigned to difficult projects to keep the development costs in line with the estimates. Subramanian and Zarnich [22] empirically illustrate that the variance in tool experience explains the variance in software development effort. Further, Subramanian and Zarnich [22] note that ICASE tool experience is critical in achieving higher productivity. Poor software productivity is attributed to the absence of ICASE tool training for systems personnel [28]. ICASE users often experience a productivity decrease for the first 3–6 months, and it often takes 12–18 months before productivity gains are visible [14]. It is amply clear from this finding that a higher level of CASE tool experience obtained over a longer period of time would result in higher productivity [22].

Kemerer [13] conjecturing the existence of learning curve in ICASE tools writes, “Integrated CASE tools have raised the stakes of the learning issue. Because these tools cover the entire life cycle, there is more to learn, and therefore the study of learning – and the learning-curve phenomenon – is becoming especially relevant”. Further, emphasizing the importance of estimating the functional form of learning curve, Kemerer [13] writes, “Managers need more information to justify their investment in CASE technology. They need a way to predict the extent of the learning curve and data to estimate its parameters so that they can determine their return on investment or similar measures for CASE tool adoption”.

The exact relationship between tool experience and development effort is not formally established. Given that exponential decreasing relationships have been observed in related software engineering applications [6,15,12], it can be argued that the relationship between development effort and tool experience is an inverse exponential relationship given by the following generalized mathematical formulation:

$$y = a_1 e^{-a_2 x},$$

where the variables y and x represent effort and tool experience, respectively. The constants $a_1 > 0$ and $a_2 > 0$ are constants that are related to the type of tool and learning rate respectively. Thus, we hypothesize the following:

Hypothesis. Increase in a programmer’s tool experience results in an exponential decrease of software effort that can be expressed in an exponential decay

form as $y = a_1 e^{-a_2 x}$. The hypothesized exponential decreasing relationship between effort and tool experience is consistent with the production function properties of convexity and variable returns to scale [1]. Banker and Slaughter [1], using COBOL programming projects, illustrated that the relationship between effort-experience is convex and variable returns to scale economies exists between effort and programmer experience/in-experience. Following Lemmas 1 and 2 illustrate the convexity of effort-experience, and effort-learning rate relationships.

Lemma 1. *Given constant $a_1 > 0$ and $a_2 > 0$, the learning curve expression $y = a_1 e^{-a_2 x}$ is decreasing and convex with respect to programmer’s tool experience.*

Proof. Taking partial derivative of the learning curve expression with respect to x , we get the following:

$$\frac{\partial y}{\partial x} = -a_1 a_2 e^{-a_2 x} < 0.$$

This shows that the relationship is decreasing. The second partial derivative is given as follows:

$$\frac{\partial^2 y}{\partial x^2} = a_1 (a_2)^2 e^{-a_2 x} > 0 \quad \forall a_1 > 0, a_2 > 0 \text{ and } x \geq 0.$$

This shows that the learning curve is convex and proves the lemma. \square

Lemma 2. *Given constant $a_1 > 0$ and $x \geq 0$, the learning curve expression $y = a_1 e^{-a_2 x}$ is convex with respect to programmer’s learning rate a_2 .*

Proof. Our proof of Lemma 2 is very similar to the proof of Lemma 1. Taking the second partial derivative of the learning curve expression with respect to a_2 we get the following:

$$\frac{\partial^2 y}{\partial a_2^2} = a_1 (x)^2 e^{-a_2 x} \geq 0 \quad \forall a_1 > 0, a_2 > 0 \text{ and } x \geq 0.$$

This proves the lemma. \square

Note: The convexity of effort-experience and effort-learning rate relationships guarantees that a programmer with higher experience and higher learning rate will have lower effort than a programmer with lower experience and lower learning rate.

4. Data, experiment and results

We use the data set used in Subramanian and Zarnich [22], Pendharkar and Subramanian [21] and Pendharkar et al. [20] studies. The data set consists of forty projects obtained from two major companies in the North Eastern USA. There were two different ICASE tools that were used in the forty projects. The first ICASE tool was Texas Instruments (TI) CASE tool IEF, and the second ICASE tool was the Electronic Data Systems (EDS) CASE tool INCASE. The projects in the data set used Rapid Application Development (RAD) and the traditional Systems Development Life Cycle (SDLC) development approaches for software development. Other variables in the data set were average programmer's ICASE tool experience, actual project effort in man months, adjusted function points (AFP), unadjusted function points (UFP), and technical complexity factor (TCF). The TCF values did not vary much across the projects [mean = 0.8923; SD = 0.1126] and projects are considered to be very similar. Interested reader is directed to Subramanian and Zarnich [22] for more information on the training data set used for this study.

Subramanian and Zarnich [22] data, while containing desirable variables for investigating the existence of learning curves, has one limitation that restricts us from using it for our experiments. The limitation is that the data set is cross-sectional, and to find the exact relationship between the ICASE tool experience and software development effort, a longitudinal data set is desirable. Longitudinal data are very difficult to obtain as most organizations do not collect the performance data to quantitatively evaluate ICASE tool learning curves [13]. Further, Kemerer [13] notes that overburdened IS staff and real life dynamics of IS projects, such as transferring team members from one project to another and staff turnover make it extremely difficult to gather data that may be used for estimating ICASE tool learning curves. Thus, in order to use Subramanian and Zarnich [13] data to estimate the learning curve between programmer's ICASE tool experience and software development effort, we make the following assumption.

Assumption: The programmers' learning curves between their ICASE tool experiences and software development effort have same functional form, and their ICASE tool experiences are similar, independent and identically distributed (iid).

In Section 6 of this paper, we analyze the impact of this assumption in detail and provide mathematical expressions and examples for estimating probability bounds for certain effort. Our assumption, while restrictive, allows us to use cross-sectional data for estimating learning curves. Also, it allows us to learn the learning curves for the team and generalize the functional form to an individual programmer; as we shall demonstrated in the following lemma.

Lemma 3. *If i th programmer's effort y_i and experience x_i has the relationship of type $y_i = a_1 e^{-a_2 x_i}$ then, under the iid assumption and for k programmers, the team effort can be approximately represented as $Y^{\text{iid}} = a_1 e^{-a_3 x}$; where x is average team ICASE tool experience and $a_3 = a_2 k$.*

Proof. Under iid assumption and k programmers in a team with ICASE tool experiences of x_1, x_2, \dots, x_k , the total project effort Y^{iid} can be given by sum of individual efforts as follows:

$$Y^{\text{iid}} = \sum_{i=1}^k y_i = a_1 [e^{-a_2 x_1} + e^{-a_2 x_2} + \dots + e^{-a_2 x_k}].$$

Assume $Y^{\text{iid}} = a_1 e^{-a_3 x}$, we have

$$a_1 e^{-a_3 \left(\frac{x_1 + x_2 + \dots + x_k}{k}\right)} = a_1 [e^{-a_2 x_1} + e^{-a_2 x_2} + \dots + e^{-a_2 x_k}].$$

After simplifying and taking natural logarithm on both sides, we have the following:

$$a_3 \left(\frac{x_1 + x_2 + \dots + x_k}{k}\right) = a_2 (x_1 + x_2 + \dots + x_k).$$

Simplifying above equation leads to $a_3 = a_2 k$. This proves the Lemma. \square

Assuming identical learning curves, we learn the relationship between development effort and average team ICASE tool experience without assuming any a priori functional form. We use artificial neural networks to learn the relationship from the data. Based on the following proposition, artificial neural networks can learn any function to any level of accuracy, and the function learned by neural network may not be considered to have any researcher bias.

Proposition 1. *An artificial neural network using sigmoid transfer function with n neurons in the input layer and one neuron in the output layer can approximate any continuous function $f : \mathfrak{R}^n \rightarrow \mathfrak{R}$ in any domain with any given accuracy.*

Proof. The reader is directed to Tromp [24] for full proof of this proposition. \square

Although an artificial neural network can learn any function to any level of accuracy, there may not be a way of knowing the type of function learnt by an artificial neural network if all the inputs are continuous. However, Pendharkar and Subramanian [21] illustrated that if the input vector contains exactly one continuous input, several other categorical inputs, and one continuous output variable then the exact pairwise function plots between the continuous input and each of the categorical inputs can be obtained.

Since our data set contains exactly one continuous input, two categorical variables and one continuous output variable, the exact function between continuous input and output variable can be elicited using the approach suggested by Pendharkar and Subramanian [21]. In order to learn the non-linear effort-experience function, we used a three layer neural network with three input nodes, seven hidden layer nodes and one output node. The number of hidden layer nodes was selected based on the popular heuristic of $2n + 1$ (where n is number of input nodes) [5]. The three inputs were software development methodology (RAD = 1, SDLC = 0), tool used (IEF = 1, INCASE = 0) and the team's tool experience in years. We chose software effort in man months as our output variable. We used all the 40 projects for learning the connection weights in a neural network. The 40 projects sample size for learning was adequate as the heuristic for learning sample size is 10 times the number of input variables [5]. After training the artificial neural network, we used an experimental design to systematically elicit the patterns that the artificial neural network had learnt.

We performed two experiments to identify team effort-experience functions for IEF ICASE tool

and INCASE ICASE tool, respectively. In each of these two experiments, we fixed the first two inputs (methodology and tool) and varied the tool experience from its initial value of 0–0.99 in intervals of 0.03. Our first experiment used SDLC methodology (methodology = 0) and IEF tool (CASE = 1) for the two fixed input values. After fixing the two inputs and varying the third input, we obtained the software effort values generated by the trained network. Fig. 1 illustrates the normalized graph of the software effort generated by the trained neural network for the different input values of tool experience. As shown in Fig. 1, when tool experience increases the software effort decreases.

For our second experiment, we kept the first two inputs fixed at RAD methodology (methodology = 1) and INCASE tool (CASE = 0) and varied the tool experience systematically. Fig. 2 illustrates the results of our second experiment. Figs. 1 and 2 illustrate the non-linear patterns that were learned by the neural network. It can be seen that both in Figs. 1 and 2, higher tool experience leads to decrease in the software effort.

After an initial visual evidence of existence of learning curve, we specifically test for the existence of hypothesized relationship. In other words, we test if increase in tool experience decreases the software effort exponentially and if the relationship can be represented in exponential decay form as follows:

$$\text{effort} = a_1 e_3^{-\text{experience} * a}$$

Using a non-linear least square regression approach, we tested our hypothesis for IEF and non-RAD, and INCASE tool and RAD methodology. Tables 1–4 show the results of our non-linear regression tests for the two experiments.

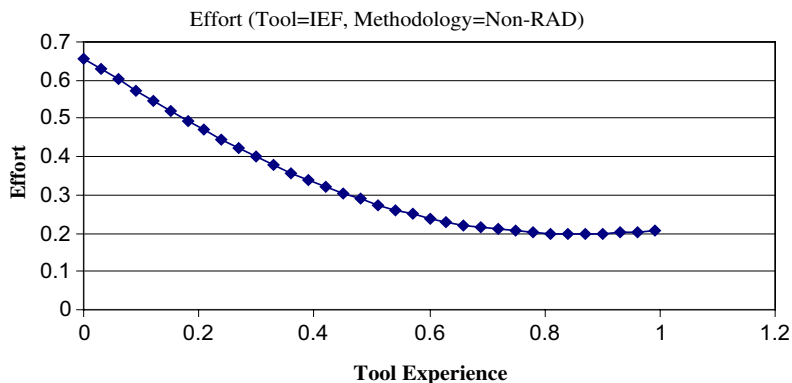


Fig. 1. Software effort vs. IEF tool experience for non-rapid application design.

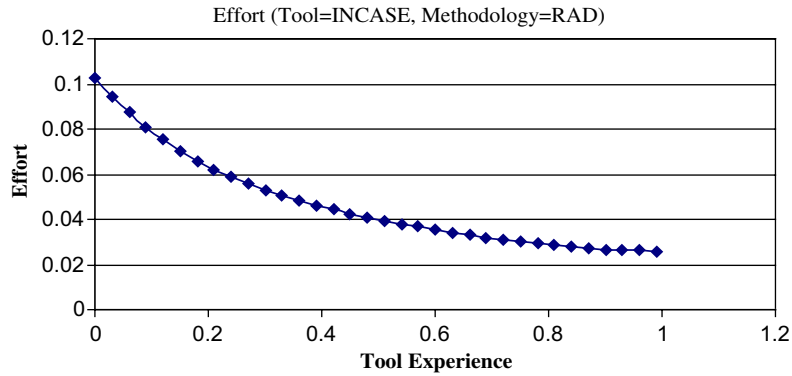


Fig. 2. Software effort vs. INCASE tool experience for rapid application design.

Table 1
The non-linear regression analysis results (IEF-NonRAD)

Source	DF	Sum of squares	Mean square	F-value	Probability (F)
Regression	1	0.6927	0.6927	1341.93	0.00001**
Error	32	0.0165	0.0005		
Total	33				

$R^2 = 0.9767$.

** Significant at $\alpha = 0.01$.

Table 2
The estimates for the constants and the t -test results (IEF-nonRAD)

Parameter	Estimate	Standard error	t -value	Probability (t)
a_1	0.6436	0.0102	62.64	0.00001**
a_3	1.5113	0.0445	33.92	0.00001**

** Significant at $\alpha = 0.01$.

Table 3
The non-linear regression analysis results (INCASE-RAD)

Source	F	Sum of squares	Mean square	F-value	Probability (F)
Regression	1	0.0145	0.0145	911.23	0.00001**
Error	32	0.0005	0.0000		
Total	33				

$R^2 = 0.9650$.

** Significant at $\alpha = 0.01$.

Table 4
The estimates for the constants and the t -test results (INCASE-RAD)

Parameter	Estimate	Standard error	t -value	Probability (t)
a_1	0.0923	0.0018	50.54	0.00001**
a_3	1.5113	0.0567	27.99	0.00001**

** Significant at $\alpha = 0.01$.

Table 5 summarizes the results of our experiments. Under the proposed iid assumption, the results indicated support for our hypothesis. Thus, we conclude that under the iid assumption learning curves exist between programmer's software development effort and his/her ICASE tool experience. The results indicate that the use of different ICASE tools may have different learning curves. For example, based on the differential equation in Table 5, we can say that IEF and SDLC have 69% learning curve (based on the coefficient of the differential equation). Whereas, INCASE and RAD have 9% learning curve. For higher learning curve percentage, increase in programmer's ICASE tool experience may lead to dramatic reduction in software effort.

Given that software development effort decreases with increase in ICASE tool experience, we investigate point elasticity of software development effort. We illustrate that the software development effort is inelastic for low ICASE tool experience (less than 0.66).

Lemma 4. *The software development effort relationship with IEF and INCASE tool experience is inelastic for low ICASE tool experience (less than 0.66).*

Table 5
The summary of the functions learnt by the ANN

Scenerio	Model learnt	Differential equation ^a
IEF and SDLC	Effort = 0.69 $e^{-1.51 * \text{experience}}$	$d(\text{effort})/d \text{exp.} = -0.69 \text{ exp.}$
INCASE and RAD	Effort = 0.09 $e^{-1.51 * \text{experience}}$	$d(\text{effort})/d \text{exp.} = -0.09 \text{ exp.}$

^a See the proof in Appendix A.

Proof. By definition of elasticity, the effort-ICASE tool experience is elastic as long as following expression holds true:

$$\frac{\text{experience} * \frac{\partial \text{effort}}{\partial \text{experience}}}{a_1 e^{-a_3 \text{experience}}} > -1.$$

After simplifying, we get the following expression:

$$-a_3 \text{experience} > -1. \quad \square$$

Since the value of a_3 , for both IEF and INCASE tools, is approximately 1.5. Substituting the value of $a_3 = 1.5$, we get experience < 0.66 for inelastic scaled software development and scaled ICASE tool experience relationship. The inelastic scaled effort-experience curve indicates that for one percent increase in scaled ICASE tool experience less than one percent drop in scaled effort is expected.

Since managers may have an option to train inexperienced programmers in a particular ICASE tool, Lemma 4 indicates that the training budget that would increase a programmer's tool experience by 1% should never exceed 1% of software development effort cost. It is important to note that the ICASE tool experience and software development effort are normalized values and not absolute values. Thus, the percentages should be interpreted carefully.

5. The impact of iid assumption and probability bounds on software development effort

In our research, we assumed that programmers' effort-experience curves are identical and programmers' experience distributions are independent and identically distributed. Given iid distribution for programmers' experience and identical learning curves, the programmer effort distribution is also iid. In real-world applications, the learning curves derived using the iid assumption may lead to errors in estimation because of the violation of the iid assumption. In this section, we provide certain insights into what impact the iid assumption may have on the true estimation of the software development effort.

First, we assume that software development effort distributions are not iid and derive an equation for the total project effort as an additive function of individual software programmer's software development efforts. Next, we show how the iid assumption impacts the total software development effort values. In the end, we derive loose and tight

upper probability bounds for software development effort being greater than a certain number, and use a central-limit theorem based probability estimator for estimating the probability that software development effort will be less than a certain number.

Assume that there are k programmers working in a software development project. Also, assume that for i th programmer the expected software development effort is given as follows:

$$y_i = \lambda_{1i} e^{-\lambda_{2i} x_i} \quad \forall i \in \{1, \dots, k\}.$$

The variables y_i , x_i , λ_{1i} , λ_{2i} are the expected software development for the i th programmer, the ICASE tool experience of the i th programmer, and two constants specific to the i th programmer. Using sum of random variables mathematical relations from the probability theory, expected software development effort Y and its variance $\text{Var}[Y]$ for the software project can be written as follows:

$$Y = \sum_{i=1}^k \lambda_{1i} e^{-\lambda_{2i} x_i}, \text{ and}$$

$$\text{Var}[Y] = \sum_{i=1}^k \text{Var}[y_i] + 2 \sum_{i=1}^{k-1} \sum_{j=i+1}^k \text{Cov}[y_i, y_j].$$

Under the iid assumption and results from Lemma 3, the expected value of the software development effort Y^{iid} can be written as follows:

$$Y^{\text{iid}} = a_1 e^{-a_2 kx}.$$

The variable x is average ICASE tool experience for the project team, the parameters a_1 and ka_2 represent the parameters learned from the historical data. The question now arises as to what is the relationship between Y and Y^{iid} ? The following lemma establishes the relationship between Y and Y^{iid} .

Lemma 5. *The estimated software development effort without iid assumption is a product of a non-negative constant ϕ and estimated software development with iid assumption. The relationship can be written as $Y = \phi Y^{\text{iid}}$, where $\phi > 0$.*

Proof. Given $Y = \sum_{i=1}^k \lambda_{1i} e^{-\lambda_{2i} x_i}$ define k constants s_i as follows:

$$s_i = \frac{e^{-\lambda_{2i} x_i}}{e^{-a_2 kx}} \quad \forall i \in \{1, \dots, k\}.$$

The expected effort can be rewritten as follows:

$$Y = e^{-a_2 kx} \sum_{i=1}^k \lambda_{1i} s_i.$$

Assume a non-negative constant ϕ defined as follows:

$$\phi = \frac{\sum_{i=1}^k \lambda_{1i} S_i}{a_1}.$$

The expected effort Y can be re-written as follows:

$$Y = \phi a_1 e^{-a_2 x} = \phi Y^{\text{iid}}.$$

This proves the lemma. \square

The value of ϕ determines how accurately Y^{iid} estimates Y . If $\phi > 1$ Y^{iid} underestimates Y ; $\phi < 1$ Y^{iid} value may be considered to be an upper bound for the estimated project effort Y . Values of ϕ that are equal or very close to one means Y^{iid} can be used to provide a reliable estimate for Y .

Under the iid assumption, it is assumed that covariances of individual effort between any two programmers will be zero. Thus, the variance of Y^{iid} will underestimate the actual variance of the project $\text{Var}[Y]$. However, a project manager may minimize the pairwise covariances between the programmers in a team by implementing effective programmers' coordination strategies. We believe that extreme programming may be one strategy to lower pairwise programmers' covariances [2].

So far, we have discussed the expected values and variances of the software development effort under non-iid and iid assumptions. We now focus on team probability density functions (pdfs) of Y and Y^{iid} . Since Y and Y^{iid} are team pdfs, which are sum of individual team member's pdf, computing team pdfs as sum of random variables, we use the moment generating functions (MGFs) [17]. Assume that $f_{Y_1}(y), \dots, f_{Y_k}(y)$ are independent probability density functions of software development efforts for each of k programmers. The MGF for total software development effort, Y can be defined as follows:

$$\varphi_Y(r) = E[e^{rY}] = \int_{-\infty}^{\infty} e^{ry} f_Y(y) dy.$$

The MGF is similar to the Fourier and Laplace transforms that are commonly used in linear systems. The primary difference between MGF and transforms used in linear systems is that the MGF is defined for real values of r . The MGF always exists for the value of $r = 0$ and provides a complete probability model of the random variable Y . Inverse transforms can be used to calculate the value of pdf from the MGF.

Using the MGF of Y , which is sum of independent random variables of k software development efforts, can be written as follows:

$$\varphi_Y(r) = \varphi_{Y_1}(r) \varphi_{Y_2}(r), \dots, \varphi_{Y_k}(r).$$

Under the iid assumption, $\varphi_{Y_i}(r) = \varphi(r)$ for all i 's. The MGF for Y^{iid} can be written as follows:

$$\varphi_{Y^{\text{iid}}}(r) = [\varphi(r)]^k.$$

The iid assumption leads to the following error in the true estimation of the MGF of k independent random software development efforts:

$$\begin{aligned} \text{Error} &= |\varphi_{Y^{\text{iid}}}(r) - \varphi(r)| \\ &= |[\varphi(r)]^k - [\varphi_{Y_1}(r) \varphi_{Y_2}(r) \cdots \varphi_{Y_k}(r)]|. \end{aligned}$$

Assume the following:

$$\begin{aligned} \varphi_{Y_{\min}}(r) &= \text{Minimum}\{\varphi_{Y_1}(r), \varphi_{Y_2}(r), \dots, \varphi_{Y_k}(r)\}, \\ \varphi_{Y_{\max}}(r) &= \text{Maximum}\{\varphi_{Y_1}(r), \varphi_{Y_2}(r), \dots, \varphi_{Y_k}(r)\}. \end{aligned}$$

The upper and lower bounds for MGF for Y^{iid} can be written as follows:

$$[\varphi_{Y_{\min}}(r)] \leq [\varphi_{Y^{\text{iid}}}(r)] \leq [\varphi_{Y_{\max}}(r)].$$

The upper and lower bounds indicate that as the difference between the MGFs of the least productive programmer in the team and most productive programmer in the team becomes low, the team effort pdf under the iid assumption may provide very reliable estimates for the actual team's development effort.

The knowledge of the effort-experience relationship may help managers to develop loose upper bounds for software development effort. The following Lemma provides a loose upper bound for the software development effort, given the effort-experience mapping function $f(x)$ and probability density function (pdf) of tool experience $g(x)$.

Lemma 6. *The probability P that software effort is greater than a certain number α has an upper bound given as follows:*

$$P[\text{effort} \geq \alpha] \leq \frac{\int_0^{\infty} f(x)g(x) dx}{\alpha}.$$

Proof. Given the team effort-experience mapping function $f(x)$, and the team probability density function of the ICASE tool experience, the expected effort $E(\text{effort})$ can be written as

$$E(\text{effort}) = \int_0^{\infty} f(x)g(x) dx.$$

From Markov inequality, we get the loose upper bound as

$$P[\text{effort} \geq \alpha] \leq \frac{E(\text{effort})}{\alpha}.$$

Substituting the value of $E(\text{effort})$ in Markov inequality, we get the result. This proves the Lemma. \square

Lemma 6 provides a general loose upper bound and can be used to estimate upper bound on both team and individual programmer software development effort. While **Lemma 6** provides a useful result, IS managers may want tighter upper bounds. Chernoff bound [27] uses MGF and provides a tight upper bound for any arbitrary random variable Y and a constant c as follows:

$$P[Y \geq c] \leq \min_{r \geq 0} e^{-rc} \varphi_Y(r).$$

The following example uses Chernoff bound to estimate the probability that the software development effort will exceed nine man-months.

Example 1: Assume that team software development effort Y is a Gaussian random variable with expected value of $\mu = 8.14$ man-months and standard deviation of $\sigma = 1$, what is the probability that software development effort of the project will exceed ten man-months?

Solution: Since Y is Gaussian random variable, the MGF of Y can be written as follows:

$$\varphi_Y(r) = e^{r\mu + r^2\sigma^2/2} = e^{8.14r + 0.5r^2}.$$

Using Chernoff bound we have,

$$P[Y \geq 10] \leq \min_{r \geq 0} e^{-10r} e^{8.14r + 0.5r^2} = \min_{r \geq 0} e^{0.5r^2 - 1.86r}.$$

We can find the minimizing r by choosing to minimize a function $f(r) = 0.5r^2 - 1.86r$. By setting the derivative of $df(r)/dr = 1r - 1.86 = 0$ yields $r = 1.86$. Applying r to the Chernoff bound yields,

$$P[Y \geq 10] \leq e^{-1.73} = 0.18.$$

Like **Lemma 6**, Chernoff bound can be used for estimating probability bounds for both team and programmer's software development effort. Also, Chernoff bound can be used for any type of pdf, Gaussian or otherwise.

The iid assumption allows us to use the central limit theorem to estimate approximate probability that software development effort Y^{iid} will be less than or equal to certain number. Using the central limit theorem, the probability for Y^{iid} being less

than a certain number, $P[Y^{\text{iid}} \leq w]$ can be written as follows:

$$P[Y^{\text{iid}} \leq w] \approx \Phi\left(\frac{w - n\mu}{\sqrt{n\sigma^2}}\right).$$

The variables Φ , n , σ , μ are standard normal distribution with mean zero and standard deviation 1, number of random variables (programmers), standard deviation of the iid variable, and the mean of iid variable. The following example illustrate the probability that total project effort is less than or equal to a certain number.

Example 2: Assume that there are five programmers working in a software development team. Also, assume that the software development effort of each programmer is independent and identically distributed with a mean of 1.5 man-month and standard deviation of 1 man-month. What is the probability that software development effort of the project will not exceed ten man-months?

Solution: Using the central limit theorem, we have

$$P[Y^{\text{iid}} \leq 10] \approx \Phi\left(\frac{10 - 5 * 1.5}{\sqrt{5}}\right) = \Phi(1.12) = 0.87.$$

In other words, there is 87% chance that the total project effort will not exceed 10 man-months.

6. Conclusion

Software engineering literature has long conjectured the existence of learning curves. Proving the existence of learning curves in software engineering was a difficult exercise because of the real-world dynamics of software engineering projects. In our research, we assumed that programmer's software development experiences were iid and the functional form of learning curves between their software development efforts and experiences were same. Using available data in the literature and artificial neural networks, we elicited team learning curves. Our results suggested that learning curve between software development effort and ICASE tool experience is convex and has a decreasing exponential relationship. Additionally, for the ICASE tools considered in this research, we found that the low programmer ICASE tool experience, the effort-experience is inelastic. The inelastic relationship suggests that improving a programmer's ICASE tool experience by 1% will reduce the software development effort by less than 1%. This seems to indicate that ICASE

tool experience is only one variable, among others, to improve productivity of software development.

After establishing the exponential decreasing relationship between software development effort and ICASE tool experience, we illustrated the impact that an iid assumption may have on true estimation of software development effort. We also derived the relationship between the software development efforts with and without iid assumption. Finally, we derived loose and tight probability upper bounds and, using certain examples, illustrated how these bounds may be useful for a software project manager.

An important aspect of our study is the production function form and decreasing variable returns-to-scale production function economy between software development effort and ICASE tool experience learning curve. The exponential decreasing relationship and convex production function indicates that non-linear scale economies exists between ICASE tool experience and software development effort. It appears that, in addition to the probability bounds discussed in this paper, techniques such as data envelopment analysis can be used to estimate inefficiencies between projects or programmers. Eliminating inefficiencies may lead to homogeneity between programmers and projects, which may lead to efforts and experiences that may closely adhere to the iid assumption. We believe that future research may focus on the application of the DEA in estimating inefficiencies in software development efforts.

The current research has demonstrated the existence of learning curves in ICASE tool environment. Another possible extension of current work may be to apply learning curve theory in integrated development environment (IDE) and perhaps, to different programming languages. We have shown that different learning rates may exist for different ICASE tools. Finch and Luebbe [10] has investigated the impact of learning rate and its variability on the project duration in manufacturing study. Future research may focus on investigating similar impact of ICASE learning rates on software development schedule.

Appendix A

Result 1: $y = ce^{-kt}$ is a general solution of differential equation $\frac{dy}{dt} = -ky$ for $k, t > 0$ and $y(0) = c$.

Proof. We prove the result by starting with the proposed differential equation:

$$\frac{dy}{dt} = -ky$$

rearranging the terms, we can rewrite the differential equation as

$$\frac{1}{y} \frac{dy}{dt} = -k$$

taking the integral of both sides, we get

$$\int \frac{dy}{y} = \int -k dt$$

after solving the above integrals, we have

$$\ln y = -kt + C$$

or,

$$y = e^{-kt} e^C$$

putting $e^C = c$, when $t = 0$, we get the result. \square

References

- [1] R.D. Banker, S.A. Slaughter, A field study of scale economies in software maintenance, *Management Science* 43 (12) (1997) 1709–1725.
- [2] K. Beck, *Extreme Programming Explained*, Addison-Wesley, Boston, 2000.
- [3] R.D. Banker, H.H. Chang, C.F. Kemerer, Evidence on economies of scale in software development, *Information and Software Technology* 36 (5) (1994) 275–282.
- [4] R.D. Banker, C.F. Kemerer, Scale economies in new software development, *IEEE Transactions on Software Engineering* 15 (10) (1989) 1199–1205.
- [5] S. Bhattacharyya, P.C. Pendharkar, Inductive, evolutionary and neural techniques for discrimination: A comparative study, *Decision Sciences* 29 (4) (1998) 871–899.
- [6] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [7] W. Chew, D. Leonard-Barton, R. Bohn, Beating murphy's law, *Sloan Management Review Spring* (1991) 5–16.
- [8] R.W. Conway, A. Schultz, The manufacturing progress function, *Journal of Industrial Engineering* (1959) 39–54.
- [9] C.E. Fauber, Use of improvement (learning) curves to predict learning costs, *Production and Inventory Management Journal Third Quarter* (1989) 57–60.
- [10] B.J. Finch, R.L. Luebbe, Risk associated with learning curve estimates, *Production and Inventory Management Journal Third Quarter* (1991) 73–76.
- [11] B.W. Foss, Fast, faster, fastest development, *Computerworld* 27 (22) (1993) 81–83.
- [12] B. Henderson-Sellers, Corrigenda: Software size estimation of object-oriented systems, *IEEE Transactions on Software Engineering* 23 (4) (1997) 260–261.
- [13] C.F. Kemerer, How the learning curve affects CASE tool adoption, *IEEE Software* 9 (5) (1992) 23–28.
- [14] J. Keyes, Gather a baseline to assess CASE impact, *Software Magazine* 10 (1990) 30–43.

- [15] L.A. Laranjeira, Software size estimation of object-oriented systems, *IEEE Transactions on Software Engineering* 1 (5) (1990) 510–522.
- [16] J. Lawler, B. Kitchenham, Measurement modeling technology, *IEEE Software* (2003) 68–75.
- [17] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, Addison-Wesley Publishing Company, Reading, MA, 1994.
- [18] J.E. Nicks, *Basic Programming Solutions of Manufacturing*, Society of Manufacturing Engineers, Dearborn, MI, 1982.
- [19] P.C. Pendharkar, An exploratory study of object-oriented software component size determinants and the application of regression tree forecasting models, *Information and Management* 42 (1) (2004) 61–73.
- [20] P.C. Pendharkar, G.H. Subramanian, J.A. Rodger, A probabilistic model for predicting software development effort, *IEEE Transactions on Software Engineering* 31 (7) (2005) 615–624.
- [21] P.C. Pendharkar, G.H. Subramanian, Mining software effort estimation functions using certain neural networks, *Proceedings of 4th INFORMS Conference on Information Systems and Technology* (1999) 220–234.
- [22] G.H. Subramanian, G.E. Zarnich, An examination of some software development effort and productivity determinants in ICASE tool projects, *Journal of Management Information Systems* 1 (4) (1996) 143–160.
- [23] K. Self, Learning curve boomerangs, *IEEE Spectrum* (1995) 17.
- [24] E. Tromp, Practical implications of a theoretical analysis of the behaviour of a multi-layer neural network, Report No. 93N063, University of Twente, Department of Engineering, 1993.
- [25] T. Wright, Factors affecting the cost of airplanes, *Journal of Aeronautical Science* 3 (1936) 122–128.
- [26] C.D. Wrigley, A.S. Dexter, A model for measuring information system size, *MIS Quarterly* 15 (2) (1991) 245–257.
- [27] R.D. Yates, D.J. Goodman, *Probability and Stochastic Processes*, John Wiley and Sons, New York, NY, 1999.
- [28] E. Yourdon, *The Decline and Fall of the American Programmer*, Prentice-Hall, Englewood Cliffs, NJ, 1988.