Learning Component Size Distributions for Software Cost Estimation: Models Based on Arithmetic and Shifted Geometric Means Rules

Shachi Sharma, *Member, IEEE,* Parag C. Pendharkar and Karmeshu

***Abstract*—Understanding software size distribution is critical to software cost estimation using COCOMO model and design of reliable production function model. This paper proposes and validates a theoretical framework based on the maximization of Shannon entropy to learn component size distribution of software systems when partial information about the moments is given. Specification of appropriate moment constraints either in the form of shifted geometric mean or arithmetic mean or both geometric and arithmetic means are considered. The models are validated using 30 real datasets. The analysis reveals that software systems where component sizes depict power-law behavior are governed by shifted geometric mean whereas those systems in which component size distribution shows exponential behavior are described by arithmetic mean. Another type of software system is also considered where the component size distribution is found to depict gamma distribution. Such systems are characterized by specification of both arithmetic and geometric means. The study underlines that the use of modern object-oriented programming languages adheres to power-law distribution indicating the existence of team synergies leading to substantial containment of software costs when compared to the use of traditional procedural programming languages.**

***Index Terms*—Characterizing moments, COCOMO model, component size distribution, exponential distribution, gamma distribution, maximum entropy principle, power-law, Shannon entropy, software cost estimation**

I. Introduction

Software engineering researchers are interested in regression functions for predicting software cost [1]. A typical regression function for software cost estimation can be written as

$$y_i = f(x_i) + \epsilon_i, \tag{1}$$

where x_i is an input vector of cost drivers or input prices and y_i is a variable measuring software cost for project $i = \{1, \ldots, N\}$, and the variable $\epsilon_i \sim N(0, \sigma^2)$. A solution to problem (1) can be obtained by solving the following convex least squares problem

$$\min_f \sum_{i=1}^{N} \left(y_i - f(x_i)\right)^2, \quad \text{s. t. } f \in \mathcal{F}, \tag{2}$$

where $\mathcal{F}$ is a family of an infinite number of functions. Generally, the problem (2) is difficult to solve [2], but if the underlying probability density function (pdf) for $f(x)$ is known then $\mathcal{F}$ can be restricted to a finite set of parametric regression functions. As an example, if $f(x)$ is known to be multivariate normal then the impact of input cost drivers on software cost is additive and linear regression method can be considered in optimizing (2). However, if $f(x)$ underlying pdf is log normal then Cobb-Douglas[1] form can be considered for $f(x)$ because the impact of cost drivers on software cost is multiplicative [3]. The primary benefit of knowing the underlying pdf is that $E(y|x)$ can be estimated consistently.

The fundamental problem of reliably estimating software cost comes down to understanding underlying software cost driver distributions. A common software cost driver is software component sizes. Software engineering literature has traditionally used exponential [4] and power-law [5] pdfs for software component sizes. Depending on which pdf is chosen, different results will be obtained. For example, the maximum likelihood (ML) estimate of unknown parameter of exponential distribution is inversely related to the arithmetic mean[2], whereas the ML estimate for unknown exponent of power-law distribution is related to geometric mean[3] [6], [7]. It is well-known that for different non-negative values, arithmetic mean (AM) is greater than geometric mean (GM). In other words, the component size aggregator for exponential distribution is arithmetic mean (additive) while for power-law distribution, it is multiplicative (geometric mean).

In addition to a reliable cost estimate, the knowledge of underlying software size distribution provides a better understanding of software cost behavior. For example, one of the widely used models is COCOMO which establishes a formal relationship between software size and software cost [8]. For a given dataset of software component sizes, under restrictive conditions, assuming same constant multiplier and constant returns to scale (CRS) relationship between software size and software cost, it can be proved that the software cost estimate

Dr. Shachi Sharma is Assistant Professor at Department of Computer Science, South Asian University, NewDelhi, 110021 India e-mail: shachi@sau.int

Dr. Parag C. Pendharkar is Professor at School of Business Administration, The Pennsylvania State University at Harrisburg, Middletown, PA 17057 United States email: pxp19@psu.edu

Dr. Karmeshu is Distinguished Professor at Department of Computer Science and Engineering, Shiv Nadar University, U.P 201314 India. Corresponding author, email: karmeshu@snu.edu.in, karmeshu@gmail.com.

Manuscript received ; revised .

[1]The COCOMO model is a univariate case of Cobb-Douglas function.

[2]In a software development project, the N component sizes x_1, x_2, ..., x_N follows exponential distribution with parameter λ_0 i.e. $f(x) = \lambda_0 e^{-\lambda_0 x}$, the maximum likelihood estimate for λ_0 is given by $\hat{\lambda_0} = \frac{N}{\sum_{i=1}^{N} x_i}$.

[3]For a power-law distribution $f(x) = Bx^{-\gamma}, \quad x \geq 1$, here B is normalization constant, the maximum likelihood estimate for parameter γ is given by $\hat{\gamma} = 1 + \frac{N}{\sum_{i=1}^{N} \log x_i}$.

for exponential component size distribution will always be higher.[4]

It is worth noting that exponential law is exhibited by Boltzmann-Gibbs distribution in which energy is considered to be a conserved variable. Using the analogy between physical system and economic system, Dragulescu and Yakovenko [9] have argued that money can be regarded as a conserved quantity while wealth can increase or decrease by way of transferring it from one agent to another. Programmer and project management skills are conserved in a project and these skills can be transferred from one software project to another when projects are faced with budget pressure [10]. Software project managers can also impact component size distributions by controlling various factors like selection of programming language, team size, team member experience, computer-aided software engineering tools, improved project management and project coordination practices [11]. The multiplicative impact of component size on software cost indicates the existence of software project management synergies. Modern object-oriented software programming languages that use code reuse, encapsulation, and polymorphism will result in smaller component sizes than older third-generation procedural languages [12]. Software complexity is also better handled using object-oriented programming languages that allow for improved module coupling [13]. The use of UML modelling allows for better programmer coordination of activities as well [14]. Furthermore, languages such as Python make extensive use of libraries (e.g., Scikit-Learn, Numpy, Pandas, etc.), which allows programmers to quickly develop applications with minimal code writing [15].

Power-law pdfs are scale-invariant. This means that power-law distributions are independent of software projects' scale and are more general and widely applicable for software projects of all sizes and budgets. The power-law distributions also adhere to the Pareto principle [16], which in software engineering context means that majority of software costs (sizes) are concentrated in a few components. The COCOMO II model uses similar cost concentration mechanisms using multiplicative software complexity factors [17]. The exponential software size distribution assumes that all components contribute towards software costs and this distribution is more aligned with the traditional COCOMO model that aggregates software sizes in an additive manner.

The understanding of software size pdfs can also provide insights into the programmer and team learning process. Performance improvement research in psychology provides evidence for individual additive skill learning and team collaborative learning processes. The performance improvement research suggests that geometric mean provides a good overall fit when collaborative learning process exists [18]–[20]. When team synergies do not exist, and individual skills play a major role then the performance improvement research suggests that additive aggregator based exponential distribution represents a better fit of the learning process [21]. The scale invariance of power-law was noticed in the learning process as well. For example, in mathematics education classroom learning process, a group of researchers found that same power-law distribution repeated themselves from learning in individuals to learning in teams to learning in a collection of teams [22].

One pertinent unexplored question relates to the study of the combined effect of both additive (characterized by arithmetic mean) and multiplicative (characterized by geometric mean) learning processes on software size distribution. The resulting effect will lead to a new software size distribution which has not so far been examined in the literature. The understanding and rationale for software size distributions will play a pivotal role in software engineering as well as software cost estimation using COCOMO model. The learning process has characteristic features which involve arithmetic and geometric means. In presence of partial information about the moments of the distributions, the question of obtaining an objective distribution is based on Maximum Entropy Principle (MEP) which not only minimizes entropy but is also consistent with the given moment constraints [23], [24]. This framework provides a theoretical justification for the emergence of various types of component size probability distributions. The two moment constraints considered in this paper are the arithmetic mean and geometric mean. We also consider a new moment constraint in the form of shifted geometric mean which is an intermediate measure between arithmetic mean and geometric mean [25]. The advantage of shifted geometric mean is that a parameter is added to all data points and its value is estimated from the real-world data. By adding this parameter, the internal validity of the results is improved, and high confidence is obtained on whether the underlying pdf is a power-law distribution. The specification of both arithmetic and geometric means allows us to consider additional distributions such as the gamma. Finally, we test and validate our approach using real-world datasets.

The paper is organized into nine sections. Section II presents maximum Shannon entropy framework and a model is constructed to provide closed form expression for component size distribution when partial information in the form of shifted geometric mean is available as a constraint. A procedure is outlined in section III based on gradient descent algorithm to estimate the parameters of the model. The real datasets are used to validate our findings in section IV. In the following section V, the exponential behavior of component size distribution is modeled when arithmetic mean is prescribed as the constraint. Section VI develops a gamma like component size distribution when both arithmetic and geometric means are specified as the constraints. Section VII discusses the implications of the study in software cost estimation of component based software development. A discussion of threats to validity of the proposed framework is presented in section VIII. The last section IX contains conclusion.

II. Component Size Distribution: Shifted Geometric Mean as Constraint

Let random variable X represent the component size in a software system with probability distribution $p_n = P(X =$

[4]The COCOMO relationship between software size (L) and development effort or cost (D) is: $D = K.L^{\delta}$, where K is constant of proportionality. Under constant return to scale assumption $\delta = 1$. This means expected development effort $E(D) = K.E(L)$, where $E(L)$ is the expected software size.

$n)$, $n = 1, 2, ..., M$ such that p_n is the probability that component size is n. The Shannon entropy of the software system is given by

$$S = -\sum_{n=1}^{M} p_n \log p_n. \tag{3}$$

It would be appropriate to consider shifted geometric mean as constraint such that

$$\sum_{n=1}^{M} \log (n+a)\ p_n = \log (Q+a) \tag{4}$$

where a is the shift parameter and Q is the geometric mean. The normalization constraint is

$$\sum_{n=1}^{M} p_n = 1. \tag{5}$$

Maximizing Shannon entropy (3) subject to shifted geometric mean constraint (4) and normalization constraint (5), results in Lagrangian function

$$\Phi = -\sum_{n=1}^{M} p_n \log p_n + \alpha \left(1 - \sum_{n=1}^{M} p_n\right) + \beta \left[\log (Q+a) - \sum_{n=1}^{M} \log (n+a)\, p_n\right]. \tag{6}$$

Differentiating (6) with respect to p_n, α and β, we have

$$-(1 + \log p_n) - \alpha - \beta \log (n+a) = 0 \tag{7}$$

$$1 - \sum_{n=1}^{M} p_n = 0 \tag{8}$$

and

$$\log (Q+a) - \sum_{n=1}^{M} \log (n+a)\ p_n = 0. \tag{9}$$

Solving (7) and (8) together results in component size distribution as

$$p_n = Z^{-1}(n+a)^{-\beta} \tag{10}$$

where $Z = \sum_{n=1}^{M} (n+a)^{-\beta}$ is the normalization constant. Note that, for large component size, (10) behaves as

$$p_n \sim n^{-\beta}, \qquad \beta > 1 \tag{11}$$

i.e. exhibiting power-law. The Lagrange's parameter β can be estimated from (9) as

$$\frac{\sum_{n=1}^{M} \log (n+a)\ (n+a)^{-\beta}}{\sum_{n=1}^{M} (n+a)^{-\beta}} = \log (Q+a). \tag{12}$$

For large M and $\beta > 1$, the summation in normalization constant Z can be approximated by integral yielding

$$Z = \sum_{n=1}^{M} (n+a)^{-\beta} \simeq \int_{1}^{M} (x+a)^{-\beta} dx. \tag{13}$$

The resultant component size probability distribution becomes

$$p(n \mid \beta, a, M) = \frac{(1-\beta)(n+a)^{-\beta}}{\left[(M+a)^{1-\beta} - (1+a)^{1-\beta}\right]}. \tag{14}$$

Algorithm 1 Gradient descent algorithm for estimating parameters β and a

Require: Empirical data
Ensure: Parameters β and a
initialize β and a
while convergence **do**
$\beta = \beta - \alpha \sum_n \left(\frac{\partial p_n}{\partial \beta}\right) \left[1 + \log \frac{p_n}{q_n}\right]$
$a = a - \alpha \sum_n \left(\frac{\partial p_n}{\partial a}\right) \left[1 + \log \frac{p_n}{q_n}\right]$
end while

The closed form of component size distribution (14) enables us to compute some other quantities of interest like mean component size and cumulative distribution.

Mean Component Size

The expected component size is given by

$$E[X] = \overline{X} = \sum_{n=1}^{M} n p_n = Z^{-1} \sum_{n=1}^{M} n(n+a)^{-\beta}. \tag{15}$$

Approximating the summation by integral in (15), the expected component size is given by

$$\overline{X} = \left(\frac{1-\beta}{2-\beta}\right) \left[\frac{(M+a)^{2-\beta} - (1+a)^{2-\beta}}{(M+a)^{1-\beta} - (1+a)^{1-\beta}}\right] - a. \tag{16}$$

Cumulative Distribution Function of the Component Size

One of the quantities of interest in software engineering is the probability that component size is less than or equal to a specified value i.e. cumulative distribution function (CDF) of the component size

$$F_X(x) = P(X \le x) = Z^{-1} \sum_{n=1}^{x} (n+a)^{-\beta}. \tag{17}$$

Again replacing summation by integral in (17) gives

$$F_X(x) = \frac{(x+a)^{1-\beta} - (1+a)^{1-\beta}}{(M+a)^{1-\beta} - (1+a)^{1-\beta}} \tag{18}$$

which shows power-law behavior for large value of x i.e.,

$$F_X(x) \sim x^{-(\beta-1)}, \qquad \beta > 2. \tag{19}$$

In order to evaluate the cumulative component size distribution (18), one needs to compute the shift parameter a and Lagrange parameter β. The procedure to compute the parameters is described in the next section.

III. Estimation of Parameters

In view of the non-linearity, it is difficult to obtain values of the parameters β and a in (12) analytically. We estimate parameters β and a using gradient descent algorithm [26] and choose popular Kullback-Leibler (KL) measure [23] as the loss function. The KL measure calculates the distance between two probability distributions. The objective is to find those values of parameters β and a which minimizes distance between $p(n)$ as computed from (14) and empirical probability distribution

TABLE I: Dataset from [5]

Package	Language	Total lines	Total components
Embedded control system I	C	1498	87
Embedded control system II	C	2709	70
Communication package	C	36019	781
Statistics package	C	16532	419
Javascript interpreter	C	27764	718
Embedded control system III	C	1373	23
C++ parser	C	43700	937
Fortran 77 parser	C	4345	90
Graphics package I	C	1635	51
Graphics package II	C	4564	71
Geophysical modeling	C	12916	46
Fortran to C converter	C	18716	502
JPEG graphics utilities	C	18485	501
C parser	C	42699	502
Abstract interpretation library	C	13816	265
Weather modeling	Fortran	15149	144
Geophysical modeling	Fortran	279638	704
Geophysical modeling	Tcl-tk	33774	394
Mobile phone calculations	Tcl-tk	5506	99
Javalin flight modeller	Tcl-tk	2593	58
C parser front end	Tcl-tk	20108	304

$q(n)$ as obtained from the dataset. The KL measure is given by

$$KL(p||q) = \sum_n p(n) \log \frac{p(n)}{q(n)}. \tag{20}$$

The gradient descent algorithm [26] estimates parameters β and a iteratively such that

$$\beta_{i+1} = \beta_i - \alpha \frac{\partial}{\partial \beta} \left[\sum_n p_n \log \frac{p_n}{q_n} \right] \tag{21}$$

and

$$a_{i+1} = a_i - \alpha \frac{\partial}{\partial a} \left[\sum_n p_n \log \frac{p_n}{q_n} \right]. \tag{22}$$

TABLE II: Details of OO software datasets

Data set	Language	Total lines	Total components
Bielak [27]	C++	70438	152
Guttag [28]	Python	511	9
Pendharkar [29]	C #	745	10
WEKA [30]	Java	29789	68
Crypto library [31]	Java	6770	25
Basic linear algebra [32]	C #	1398	31
Codec library [33]	Java	10543	58

Here, α is the learning rate. The equations (21) and (22) can be further simplified to

$$\beta_{i+1} = \beta_i - \alpha \sum_n \left(\frac{\partial p_n}{\partial \beta} \right) \left[1 + \log \frac{p_n}{q_n} \right] \tag{23}$$

and

$$a_{i+1} = a_i - \alpha \sum_n \left(\frac{\partial p_n}{\partial a} \right) \left[1 + \log \frac{p_n}{q_n} \right]. \tag{24}$$

The gradient descent algorithm is presented in Algorithm 1. The partial derivatives of p_n with respect to β and a, as obtained from (14) are

$$\begin{aligned} \frac{\partial p_n}{\partial \beta} = & \frac{1}{\left\{ (M+a)^{1-\beta} - (1+a)^{1-\beta} \right\}^2} \\ & \Big[\left\{ (M+a)^{1-\beta} - (1+a)^{1-\beta} \right\} \\ & \left\{ (\beta-1)(n+a)^{-\beta} \log(n+a) - (n+a)^{-\beta} \right\} \\ & - \left\{ (1-\beta)(n+a)^{-\beta} \right\} \left\{ (1+a)^{1-\beta} \log(1+a) \right. \\ & \left. - (M+a)^{1-\beta} \log(M+a) \right\} \Big] \end{aligned} \tag{25}$$

and

$$\begin{aligned} \frac{\partial p_n}{\partial a} = & \frac{(1-\beta)}{\left\{ (M+a)^{1-\beta} - (1+a)^{1-\beta} \right\}^2} \Big[\left\{ (M+a)^{1-\beta} \right. \\ & \left. - (1+a)^{1-\beta} \right\} \left\{ -\beta (n+a)^{-\beta-1} \right\} \\ & - \left\{ (1-\beta)(n+a)^{-\beta} \right\} \left\{ (M+a)^{-\beta} - (1+a)^{-\beta} \right\} \Big]. \end{aligned} \tag{26}$$

Using algorithm 1, the statistical procedure to validate model for real datasets is outlined in the next section.

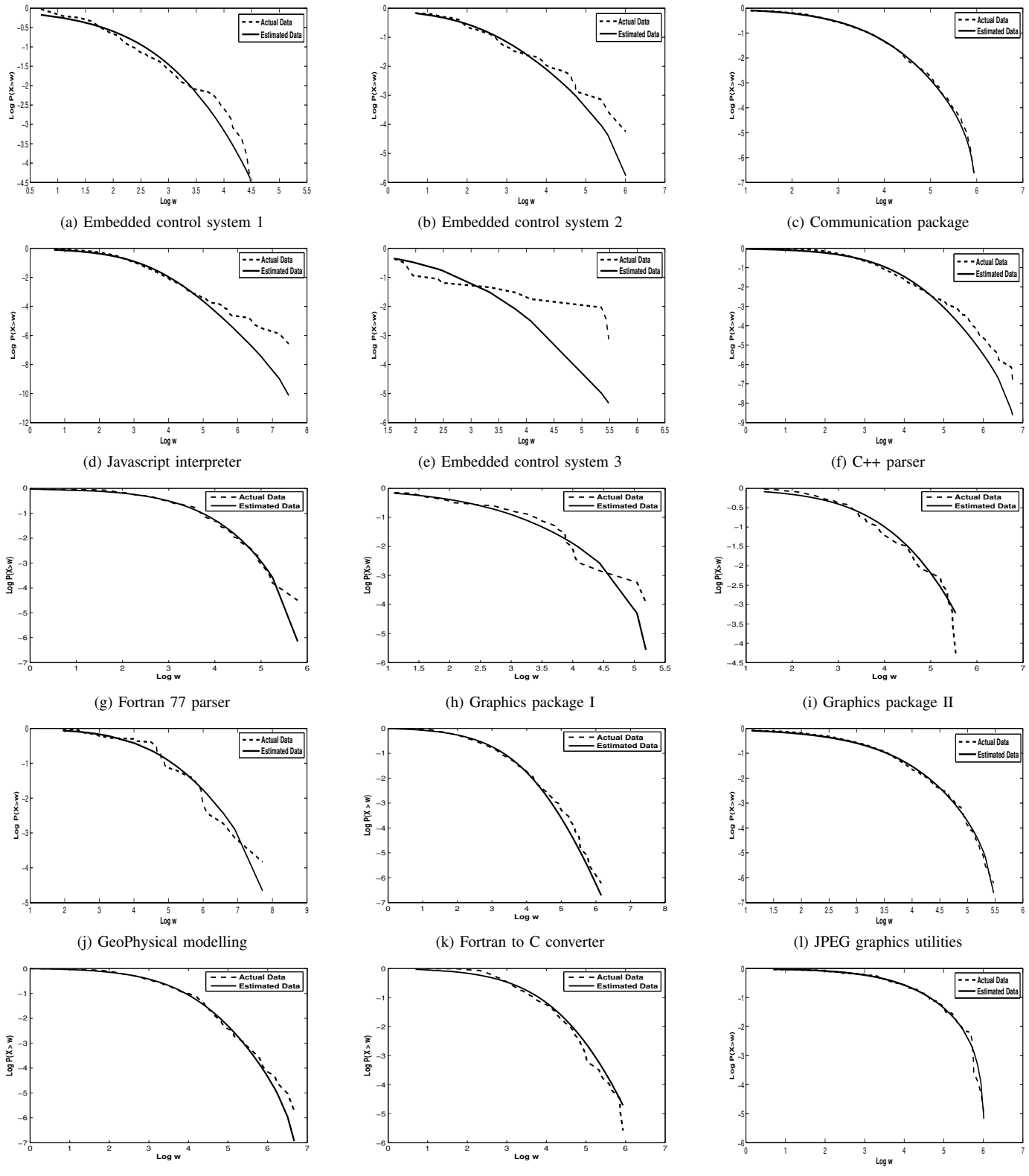
IV. Results of Experiments

To validate the MEP based component size distribution as presented in section II, we conduct experiments using many real datasets. Highlighting the importance of component

TABLE III: Estimated values of parameters using gradient descent based procedure

Package or dataset	Maximum component size M	Estimated shift parameter a	Estimated power exponent parameter β	KL measure
Embedded control system I	175	45.0056	4.9261	0.0002
Embedded control system II	511	14.0606	2.2846	0.0005
Communication package	410	110.0176	4.3599	0.0042
Statistics package	228	700.0715	17.0773	0.0012
Javascript interpreter	2228	47.0130	3.7136	0.0525
Embedded control system III	454	30.0510	3.2866	0.00001
C++ parser	965	99.0203	4.3276	0.0098
Fortran 77 parser	365	173.0103	5.5894	0.0006
Graphics package I	191	28.0210	2.5666	0.0016
Graphics package II	542	125.0062	3.6899	0.0009
Geophysical modeling C	3036	137.0069	2.2402	0.0012
Fortran to C converter	1193	80.0198	4.4726	0.0014
JPEG graphics utilities	254	280.0059	9.7986	0.0001
C parser	2160	150.0441	4.0335	0.1835
Abstract interpretation library	846	125.0168	4.3223	0.0685
Weather modeling	422	108.0115	2.1365	0.0493
Geophysical modeling Fortran	2430	5000	14.2803	0.0058
Geophysical modeling Tcl-tk	2252	57.0889	2.6205	0.0530
Mobile phone calculations	465	53.1012	2.6786	0.0591
Javalin flight modeller	253	27.1132	2.0760	0.0952
C parser front end	913	115.0293	3.7954	0.1345
Bielak	2924	250.0115	4.0412	0.0552
Guttag	227	20.0394	2.4298	0.0001
Pendharkar	473	45.0403	3.1069	0.0001
WEKA	2524	1000	3.8349	0.0082
Crypto library	1145	450.0134	2.7409	0.0030
Basic linear algebra	141	43.1618	1.5213	0.0003
Codec library	1022	97.0102	2.1000	0.0138

size distribution in softwares, Hatton [5] employs statistical mechanics approach to develop analytical model to capture power-law distribution being exhibited by softwares developed in different programming languages. The dataset used in his study are shown in Table I. Also, some other datasets from object-oriented programming languages such as C++, Java, Python and C# are chosen to test the MEP based component size distribution. These datasets are described in Table II. The C++ dataset used in our study already had software size data for different components. For Java, Python and C# datasets, a component was defined either as a model or a service. The WEKA dataset has been constructed from 11 packages from WEKA repository, and model components for this dataset sometimes spanned multiple classes. Since C# programming implemented only one model (Ant colony algorithm), service components sizes are used for this application. A service component is an element of a program that can be deployed independently offering a predefined service and can communicate with other components [34]. To improve the internal validity of data collection, separate authors transcribed model component and service component data. The parameters β and a are estimated



(a) Embedded control system 1 (b) Embedded control system 2 (c) Communication package

(d) Javascript interpreter (e) Embedded control system 3 (f) C++ parser

(g) Fortran 77 parser (h) Graphics package I (i) Graphics package II

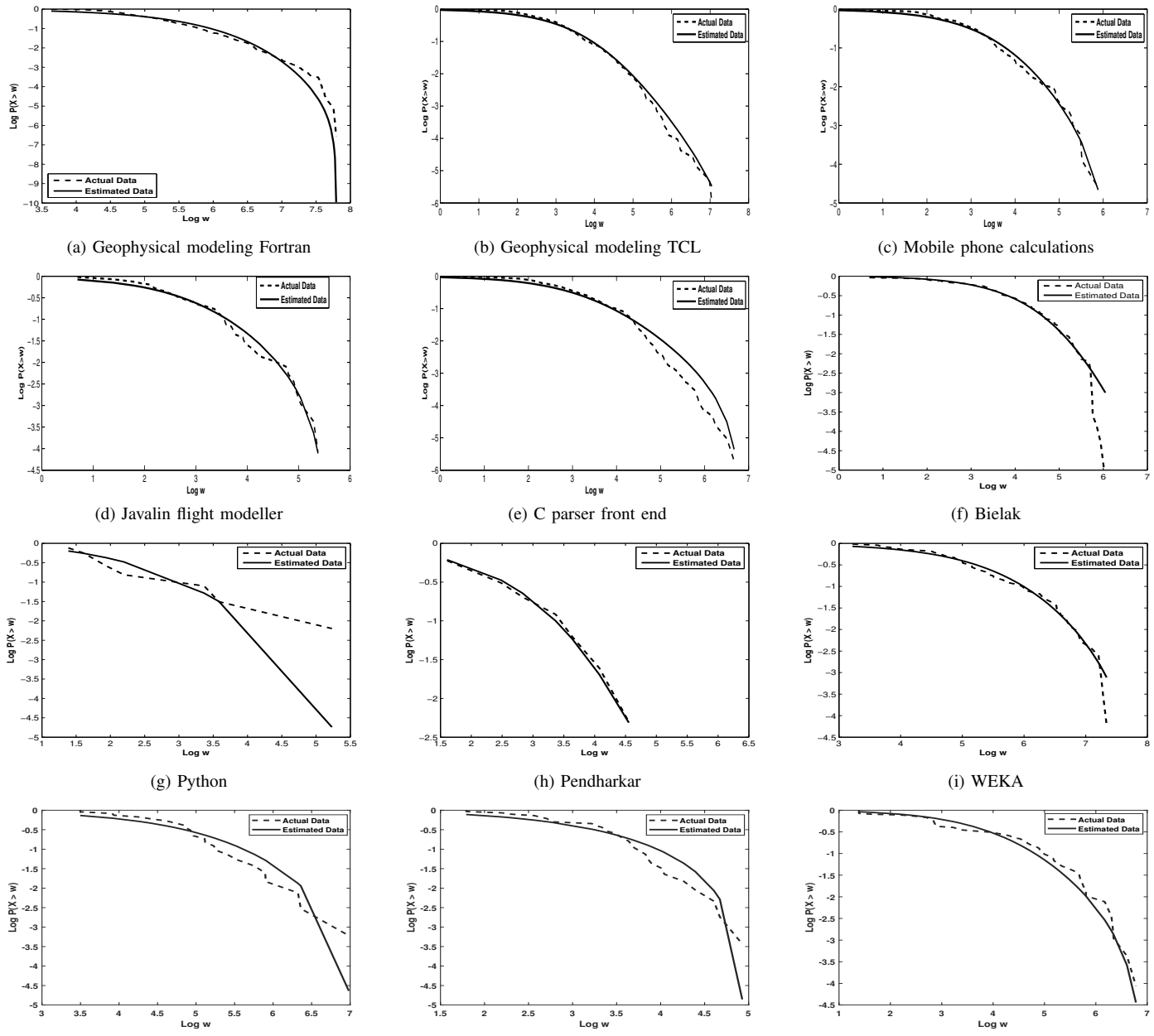
(j) GeoPhysical modelling (k) Fortran to C converter (l) JPEG graphics utilities

(m) C Parser (n) Abstract interpretation library (o) Weather modeling

Fig. 1: Comparison of complementary cumulative component size distribution

using Algorithm 1 and test whether the cumulative component size distribution (18) conforms to the empirical distribution as obtained from the data is performed. To this end, we construct,

Null Hypothesis H_0: The Cumulative component size distribution (18) fits well to the empirical cumulative component size distribution i.e. for all x,



(a) Geophysical modeling Fortran (b) Geophysical modeling TCL (c) Mobile phone calculations

(d) Javalin flight modeller (e) C parser front end (f) Bielak

(g) Python (h) Pendharkar (i) WEKA

(j) Crypto library (k) Basic linear algebra (l) Codec library

Fig. 2: Comparison of complementary cumulative component size distribution (continuation from Fig. 1)

$$F_X(x) = F_0(x)$$

Alternative Hypothesis H_1: there exists x such that

$$F_X(x) \neq F_0(x).$$

For testing goodness of fit, we use Kolmogorov-Smirov (KS) statistic [35]. Besides the value of KS statistic, Clauset *et al.* [7] emphasize on the significance of p value while fitting power-law distributions to empirical data. If p value is large and close to 1, it means that the null hypothesis is retained. If p value is small then the null hypothesis is not supported by the dataset indicating significant difference between fitted cumulative component size distribution and empirical cumulative component size distribution.

The values of the parameters β and a for datasets of Table I and Table II are presented in Table III. The results of KS test are shown in Table IV. The fitted cumulative component size distributions are found to be in close agreement with empirical distributions. Both the actual and fitted complementary cumulative component size distribution are plotted in Figures 1 and 2 for valid KS test.

An important observation from the results of Table III and Table IV is that the value of shift parameter turns out to be very large for the cases where null hypothesis is rejected. For large value of shift parameter a, it can be easily shown that shifted geometric mean approximates arithmetic mean. Rewriting (4), we have

TABLE IV: Results of KS test

Package	KS value	p value	Null hypothesis accepted
Embedded control system I	0.1343	0.8881	Yes
Embedded control system II	0.0564	1.0000	Yes
Communication package	0.0488	0.9880	Yes
Statistics package	0.2209	0.0252	No
Javascript interpreter	0.0840	0.7772	Yes
Embedded control system III	0.2276	0.8709	Yes
C++ parser	0.0851	0.5625	Yes
Fortran 77 parser	0.0566	1.0000	Yes
Graphics package I	0.0809	0.9999	Yes
Graphics package II	0.0743	0.9986	Yes
Geophysical modeling C	0.1081	0.9750	Yes
Fortran to C converter	0.0769	0.8660	Yes
JPEG graphics utilities	0.0500	0.9977	Yes
C parser	0.0756	0.6931	Yes
Abstract interpretation library	0.0667	0.9688	Yes
Weather modeling	0.0485	0.9996	Yes
Geophysical modeling Fortran	0.0771	0.1389	Yes
Geophysical modeling Tcl-tk	0.0530	0.9811	Yes
Mobile phone calculations	0.0781	0.9865	Yes
Javalin flight modeller	0.0952	0.9874	Yes
C parser front end	0.0455	0.9989	Yes
Bielak	0.0777	0.9037	Yes
Guttag	0.2500	0.9290	Yes
Pendharkar	0.1250	1.0000	Yes
WEKA	0.0746	0.9897	Yes
Crypto library	0.1265	0.9776	Yes
Basic linear algebra	0.1446	0.9097	Yes
Codec library	0.1371	0.7258	Yes

$$\sum_{n=1}^{M} \log\left[a\left(1+\frac{n}{a}\right)\right] p_n = \log\left[a\left(1+\frac{Q}{a}\right)\right]. \tag{27}$$

We get on simplification

$$\sum_{n=1}^{M} \log\left(1+\frac{n}{a}\right) p_n = \log\left(1+\frac{Q}{a}\right). \tag{28}$$

If the shift parameter is very large, i.e. $n << a$, then

$$\log\left(1+\frac{n}{a}\right) = \frac{n}{a} + \left(\frac{n}{a}\right)^2 + ...$$

TABLE V: Results: arithmetic mean as constraint

Dataset	Parameter λ	KS value	p value	Null hypothesis accepted
Withrow [36]	0.0016	0.3224	0.7091	Yes
Basili [37]	0.0058	0.1323	1.0000	Yes
Statistics Package [5]	0.0245	0.2698	0.0029	No

and

$$\log\left(1+\frac{Q}{a}\right) = \frac{Q}{a} + \left(\frac{Q}{a}\right)^2 + ...$$

Retaining the first dominant term and neglecting second and higher order terms, we find, for large a, both arithmetic and geometric means become equal i.e.

$$\sum_{n=1}^{M} n p_n = Q = A. \tag{29}$$

Accordingly, in the following section, we explore the case when the Shannon Entropy (3) is maximized using arithmetic mean as the constraint.

V. Arithmetic Mean Specified

Noting that A denotes the arithmetic mean of the component sizes in a given software system, maximization of (3) subject to arithmetic mean

$$\sum_{n=1}^{M} n\ p_n = A \tag{30}$$

and the normalization (5) constraints results in exponential component size distribution

$$p_n = \left[\frac{1-e^{-\lambda}}{e^{-\lambda}\left(1-e^{-M\lambda}\right)}\right] e^{-n\lambda}, \qquad n = 1,\ 2,\ ...,\ M \tag{31}$$

where λ is the Lagrange's parameter which can be determined from constraint (30) i.e.,

$$A = \frac{1}{1-e^{-\lambda}} - \frac{M e^{-M\lambda}}{1-e^{-M\lambda}}. \tag{32}$$

Cumulative Distribution of Component Size

The cumulative distribution function of the component size in this case is given by

$$F_X(x) = \frac{1-e^{-\lambda x}}{1-e^{-M\lambda}}. \tag{33}$$

The above results are validated for Withrow dataset [36] which gives component sizes in Ada software, Basili dataset [37] from a large scale project in Fortran language and statistics package of Table I. Using the data, the model parameter λ is computed from (32). The cumulative distribution of component size is calculated from (33) and KS test is performed for validation. The results are shown in Table V and the KS test passes for Withrow [36] and Basili [37] datasets. The comparative graphs are presented in Figures 3 and 4.

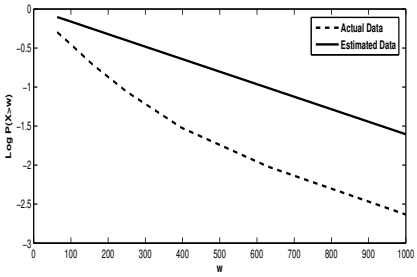


Fig. 3: Complementary CDF plot of Withrow dataset [36].

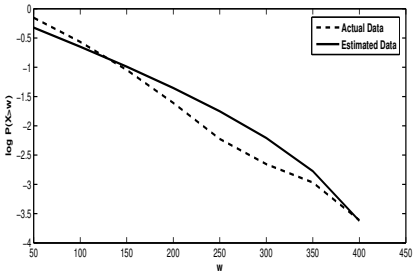


Fig. 4: Complementary CDF plot of Basili dataset [37].

TABLE VI: Results: arithmetic and geometric mean together as constraints

Dataset	μ	θ	KS value	p value	Null hypothesis accepted
Withrow [36]	0.6467	0.0015	0.1364	1,0000	Yes
Basili [37]	2.1160	0.0155	0.0179	1.0000	Yes
Statistics Package [5]	2.3409	0.0593	0.1794	0.1119	Yes

However, for statistics package of Table I, we further explore the maximization of Shannon entropy (3) with both arithmetic and geometric mean as constraints in the following section.

VI. Both Arithmetic and Geometric Mean Specified

The Shannon entropy

$$S = -\sum_{n=1}^{M} p_n \log p_n \tag{34}$$

is maximized subject to constraints of arithmetic mean

$$\sum_{n=1}^{M} n\ p_n = A, \tag{35}$$

geometric mean

$$\sum_{n=1}^{M} (\log n)\ \ p_n = \log Q \tag{36}$$

and normalization

$$\sum_{n=1}^{M} p_n = 1. \tag{37}$$

Following the same procedure as illustrated in section II, the component size distribution turns out to be

$$p_n = \frac{\lambda_1^{1-\lambda_2}\ n^{-\lambda_2}\ e^{-n\lambda_1}}{\Gamma(1-\lambda_2) - \Gamma(1-\lambda_2, (M-1)\lambda_1)} \tag{38}$$

where λ_1 and λ_2 are Lagrange's parameters corresponding to arithmetic mean and geometric mean constraints respectively, $\Gamma(s,x)$ is the incomplete gamma function. For large M with $\theta = \lambda_1$ and $\mu = 1 - \lambda_2$ and using the identity 6.5.32 of [38], $\Gamma(s,x) \to 1, \text{as } x \to \infty$, (38) is approximated by gamma distribution i.e.

$$p_n(\theta, \mu) = \frac{\theta^{\mu}\ n^{\mu-1}\ e^{-n\theta}}{\Gamma(\mu)}, \quad n > 0. \tag{39}$$

The gamma distribution (39) is again tested for Withrow [36], Basili [37] datasets, and statistics package of Table I. The results are shown in Table VI. It is worth noting that the KS test passes for statistics package of Table I. For Withrow and Basili datasets, the KS value is found to be less than that for the case when only arithmetic mean is considered. Also, the p value turns out to be unity for Withrow dataset. In this case, gamma distribution is a better fit than exponential distribution.

The implications of this study are discussed in the next section.

VII. Discussion

Once the pdf of software size distribution is known, it can be used to obtain the probability distribution of software cost using COCOMO model. However, to get an inkling about the impact of expected software size on total software cost, we use the the relationship between expected component size, $E(X)$, and software size is $L = N \times E(X)$. The expected cost in the COCOMO model can be written as $K[N \times E(X)]^{\delta}$. Using the expected component sizes for power-law distribution (16) and exponential distribution (32), we get expected costs, for large M, as $KN^{\delta}M^{\delta}$ and $KN^{\delta}(1-e^{-\lambda})^{-\delta}$ respectively. These expressions suggest that software cost will grow as the number of components increases. However, the consequence of the long-tail of power-law distribution is that bulk of the contribution towards software size comes from large number of small size components exceeding the contribution of a few very large sized components [6]. Thus, it becomes more economical to procure these large sized components from third parties rather than developing them in house. Ideally, managers should aim for a higher value of power-law exponent β and a lower value of maximum component size M to lower overall software development cost. When managers attempt both then they are trying to reduce variance in component sizes to contain software cost for a medium to large size software project (i.e., $\beta > 2$). Since $\frac{N}{\lambda_0} = \sum_{i=1}^{N} x_i$, the growth of software development cost for an exponential component size distribution, in big-O notation, can be written as $O\left(\left(\sum_{i=1}^{N} x_i\right)^{\delta}\right)$. The cost containment in exponentially distributed component sizes comes down to reducing the total size of components. While the primary focus of this study is to better understand software size distribution so that software

cost can be predicted reliably, (1) can be easily extended to the following production function form [39]

$$y_i = f(x_i) + v_i - u_i \tag{40}$$

where v_i is statistical noise and is assumed to be independent and identically distributed with a two-sided normal distribution, and u_i defines cost inefficiency that shows how far a firm operates above the cost frontier. The dual of (40) forms a stochastic cost function, which can be used for stochastic cost frontier analysis [39]. To the best of our knowledge, stochastic cost frontier analysis is not used in software engineering literature and using such analysis with reference to software size distributions is a worthwhile undertaking to gain insights into software cost drivers. The primary usefulness of such analysis is that after a software development project is complete, components can be ranked in terms of their cost inefficiencies and software development processes for components lying on cost efficiency frontier can be analyzed as role models.

The returns to scale (RTS) relationship between software size and software cost is an unsettled question in software engineering literature [1]. There are two primary techniques used for testing RTS relationships – data envelopment analysis (DEA) and stochastic frontier analysis (SFA) – and both of these techniques yield conflicting results when used on the same datasets. The proponents of SFA argue that common production process assumption is seldom satisfied in software engineering datasets and SFA is a more important technique to use [1]. The proponents of DEA argue that SFA imposes a predetermined untested production function relationship on software engineering datasets and non-parametric DEA models do not impose any such relationship [40]. The software size distributions derived in this paper may be used in part to explain the extent to which common production process assumption is satisfied.

When component sizes follow power-law distribution, software cost is primarily governed by the total number of components. The project managers are encouraged to use fewer components albeit the size of components may be large implying either more number of classes or large size classes in a component. Accordingly, single responsibility principle [41] must be followed while designing classes so that the quality and reliability of the software will not be compromised [42]. Further, the classification of components is also critical and component cohesion needs to be considered in determining component sizes as Lack of Cohesion in Method (LCOM) induces fault-proneness [43]. Generally, GUI components are cohesive, but model components that span a single class or multiple classes, cohesion plays an important role. Large size incohesive classes are considered unnecessarily complex and should be avoided, but large size cohesive classes[5] should be preferred because they are easier to develop, reuse, and maintain [43]. When component sizes follow an exponential distribution, software costs are mostly determined by the total software size. Managing such projects may be challenging because all components are equally important for the overall functionality of a software application. When gamma fits distribution of component size, it represents a characteristic of projects where programmers with identical skills are assigned software development responsibilities with different levels of complexity [44].

During early phases of software development where software managers have a choice of selecting a programming language, software tools and programmers; this study suggests that modern object-oriented and end-user programming languages must be chosen because the component size distribution for these programming languages has a higher probability of fitting the power-law distribution. Stevanetic and Zdun [45] argue that the creation of software components that are understandable is key to improve the quality of component models. Pan *et al.* [46] indicate that cohesive classes are better as they improve software stability and reduce ripple effects caused during software maintenance. Li *et al.* [47] argue that software reliability can be improved by identifying influential modules in the early stages of software development. When the results of our study are combined with some of the studies in the literature, it appears that power-law software size distributions, better understanding and planning of cohesive software components early in the software development phase are key to create low cost, high quality, stable and maintainable software.

VIII. Threats to Validity

We have proposed a unified analytical framework based on MEP which requires specification of moments to obtain various component size distributions. In this regard, using shifted geometric mean yields component size distribution which mimics asymptotically power-law behavior, while arithmetic mean and both arithmetic mean in conjunction with geometric mean lead to exponential and gamma distributions, respectively. It is interesting to point out that MEP is a very general approach which can work across datasets from diverse projects and results depend upon the choice of characterizing moments. In this section, we discuss issues related to the validity of proposed framework.

A. Internal Validity

The major threat to internal validity of the proposed framework is when moments constraints have to be modified in light of availability of new datasets. Such a situation may arise when component sizes do not conform to the distributions considered in this paper. Accordingly, new constraints need to be identified that may capture the empirically observed distribution.

Another aspect related to internal validity deals with the underlying stochastic dynamics of the software development process which in equilibrium yields desired component size distribution. As pointed out by Whitt [48], "*However, even when equilibrium is attained, in general there can be many stochastic processes having the same equilibrium distribution. In addition to the equilibrium distribution, it is useful to describe the fluctuations or transient behavior of the stochastic*

[5]A cohesive class is a class where its methods and variables are related to each other.

process". This issue can be appreciated in light of the discussion by Simon [49] where he analyses a class of distributions in a wide range of empirical data corresponding to diverse phenomena. Further, Simon notes that any similarity in these diverse phenomena can be attributed to the common property of the probabilistic mechanism. It is worth highlighting that the similar observations have been made by Concas *et al.* [50] about the underlying stochastic generation mechanisms involved in software development process.

B. External Validity

External validity is concerned with the applicability of the proposed framework to other software systems. The functional relatedness of the elements of a module, referred to as cohesion, can be extended to deal with the forces that cause a module or class to change [41]. As pointed in section IV, a component is a unit of functionality which is a collection of modules or classes. Therefore, adherence to single responsibility principle is one of the challenging tasks which in turn becomes a plausible threats of external validity.

As noted by Concas *et al.* [50], the software development process involves numerous decisions which can suitably be modeled as a stochastic process. The underlying stochastic mechanisms have the potential to explain the causal relationships between quality of software and various factors like team dynamics, programming language, software engineering tools, project management and coordination practices etc. Any static or dynamic change in these factors may pose a threat to external validity of the proposed framework. These issues require broadening of the framework to incorporate underlying stochastic dynamics or stochastic variables of interest in the problem formulation. This is likely to open a new modeling paradigm for software engineering.

IX. Conclusion

Software size is a major independent variable in COCOMO and COCOMO II models. This study uses a mathematical framework to maximize Shannon's entropy to learn software component size distributions. Shifted geometric, geometric and/or arithmetic means along with normalization constraint were used to learn power-law, exponential and gamma component size distributions. The results of the study indicate that as programming languages evolved, the behavior of underlying component size distributions changed as well. For object-oriented programming languages (C++, C#, Java and Python) the component size distributions are described by power-law. For recent end-user programming languages (e.g., Python), power-law distributions can also be used to describe component sizes. Generally, we believe that the results of our study are generalizable because of diversity of programming languages and software component types (GUI, Model, Service components among others) used in the study, but there is still a little bit of bias of procedural programming languages.

Among the secondary conclusions of this study is the desirability of developing CD type software cost forecasting models. The traditional COCOMO model has the Cobb-Douglas form. Pendharkar *et al.* [51] illustrated that other variables such as team size can also be added to the traditional COCOMO model to extend and improve its forecasting. The benefit of using CD type software cost functions is that these functions can be used for stochastic cost analysis (SCA). The SCA analysis allows use of technical efficiency and random noise that is traditionally ignored in software analysis. The study also suggests that the software component size distribution type may be used as a discriminating parameter for software engineering research. Software engineering researchers have used ISBSG (isbsg.org) datasets to test their hypotheses [11]. DEA research in software engineering [52] often uses different software projects for software project productivity analysis [53]. Most of these studies use the entire dataset to draw their conclusions. This study suggests that, wherever possible, software component size distribution may be used to screen out projects where software size distributions are dissimilar. By using software projects where software size distributions are similar, more robust research conclusions can be drawn.

Highlighting the importance of component size distribution in softwares, Hatton [5] employs Boltzmann-Gibbs statistical mechanics framework to formulate analytical model to capture power law distribution being exhibited by softwares developed in different languages. It is worth noting that Tsallis proposed a non-extensive statistical mechanics framework to generate power law behavior in a wide variety of systems [54]. The apparent success of non-extensive statistical mechanics, in contrast to Boltzmann-Gibbs mechanics, suggest a deeper question posed by Gell-mann and Tsallis [54], "*An intriguing that remains unanswered is: exactly what do all these systems have in common? One suspects, of course, that the deep explanation must arise from microscopic dynamics. The various cases could all be associated with something like a scale-free dynamical occupancy of phase space, but this certainly deserves further investigation*". Motivated by this, several investigations have been carried out including in communication network. Karmeshu and Shachi [55]–[58] have examined the emergence of power law in networks using MEP framework based on Tsallis entropy. The success of non-extensive mechanics would be useful to explore the possibility of its application in the analysis of software systems.

X. Acknowledgement

The authors thank the anonymous reviewers for providing useful comments and suggestions that helped to improve the work.

References

[1] B. A. Kitchenham, "The question of scale economies in software– Why cannot researchers agree?", Information and Software Technology, vol. 44, no. 1, pp. 13—24, 2002.
[2] D. L. Hanson and G. Pledger, "Consistency in concave regression", Annals of Statistics, vol.4, no. 6, pp. 1038–1050, 1976.
[3] A. S. Goldberger, "The interpretation and estimation of Cobb-Douglas functions", Econometrica, vol. 35, no. 3-4, pp. 464–472, 1968.
[4] Y. K. Malaiya and J. Denton, "Module size distribution and defect density", Proceedings of 11^{th} international symposium on software reliability engineering, ISSRE, 2000.
[5] L. Halton, "Power-law distributions of component size in general software systems", IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 566–572, 2009.

[6] M. E. Newman, "Power laws, Pareto distributions and Zipf's law", Contemporary Physics, vol. 46, no. 5, pp. 323–351, 2005.

[7] A. Clauset, C. R. Shallz and M. E. J. Newman, "Power-law distributions in empirical data", SIAM Review, vol. 51, no. 4, pp. 661–703, 2009.

[8] A. A. Abdulmajeed, M. A. Al-jawaherry and T. M. Tawfeeq, "Predict the required cost to develop software engineering projects by using machine learning", Journal of Physics: Conference Series, vol. 1897, 2021.

[9] A. Dragulescu and V. M. Yakovenko, "Exponential and power-law probability distributions of wealth and income in the United Kingdom and the United States", Physica A, vol. 299, pp. 213–221, 2001

[10] N. Nan and D. E. Harter, "Impact of budget and schedule pressure on software development cycle, time and effort", IEEE Transactions on Software Engineering, vol. 35, no. 5, pp. 624–637, 2009.

[11] P. C. Pendharkar and J. A. Rodger, "An empirical study of the impact of team size on software development effort", Information Technology and Management, vol. 8, pp. 253–262, 2007.

[12] R. S. Sreekumar and R. V. Sivabalan, "A survival study of object oriented principles on software project development", 2015 Global Conference on Communication Technologies (GCCT), pp. 307–310, 2015.

[13] L. C. Briand, J. W. Daly, and J. K. Wust,"A unified framework for coupling measurement in object-oriented systems", IEEE Transactions on Software Engineering, vol. 25, no. 1, pp. 91–121, 1991.

[14] K-H Doan and M. Gogolla, "Quality improvement for UML and OCL models through bad smell and metrics definition", 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 774–778, 2019.

[15] W. Jordan, A. Bejo and A. G. Persada, "The Development of lexer and parser as parts of compiler for GAMA32 processor's instruction-set using Python", 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), pp. 450–455, 2019.

[16] V. Pareto and A. N. Page, *Manuale di economia politica ("Manual of political economy")*, A.M. Kelley, ISBN 978-0-678-00881-2, 1971.

[17] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy and D. J. Reifer, *Software Cost Estimation with COCOMO II*, Prentice Hall Inc., Upper Saddle River, NJ, 2000.

[18] J. A. Fadul, "Collective learning: applying distributed cognition for collective intelligence", The International Journal of Learning, vol. 16, no. 4, pp. 211–220, 2009.

[19] G. Madirolas and G. G. Polavieja, "Wisdom of the confident: using social interactions to eliminate the bias in wisdom of the crowds", Collective Intelligence, pp. 1–4, 2014.

[20] C. C. Loannou, G. Madirolas, F. S. Brammer and H. A. Rapley, "Adolescents show collective intelligence which can be driven by a geometric mean rule of thumb", PLOS One, pp. 1–17, 2018.

[21] A. Heathcote and S. Brown, "The power law repealed: The case for an exponential law of practice", Psychonomic Bulleetin and Review, vol. 7, no. 2, pp. 185–207, 2000.

[22] T. Hills, A. C. Hurford, W. M. Stroup and R. Lesh, "Formalizing learning as a complex system: Scale invariant power-law distributions in group and individual decision making", in *Foundations for the Future in Mathematics Education*, R. A. Lesh, E. Hamilton and Kaput (eds.), Lawrence Erlbaum Associates, Mahwah, NJ, pp. 225–244, 2007.

[23] Karmeshu (Ed.), *Entropy Measures, Maximum Entropy Principle and Emerging Applications*, Springer-Verlag Berlin Heidelberg, 2003.

[24] J. Peterson, P. D. Dixit and K. A. Dill, "A maximum entropy framework for nonexponential distributions", PNAS, vol. 110, no. 51, pp. 20380-–20385.

[25] W. Cook, T. Koch, D. E. Steffy and K. Wolter, "An exact rational mixed-integer programming solver", International Conference on Integer Programming and Combinatorial Optimization, pp. 104–116, 2011.

[26] D. P. Bertsekas and M. A. Belmont, *Nonlinear Programming*, Athenas Scientific, 1995.

[27] J. Bielak, "Improving size estimates using historical data", IEEE Software, vol. 17, no. 6, pp. 27-–35, 2000.

[28] J. Guttag, *Introduction to computation and programming using Python*, 2nd edition, The MIT Press, Cambridge, MA, 2016.

[29] P. C. Pendharkar, "An ant colony optimization heuristic for constrained task allocation problem", Journal of Computational Science, vol. 7, pp. 37–47, 2015.

[30] WEKA, https://www.cs.waikato.ac.nz/ml/weka/

[31] Apache Commons Crypto Library, https://commons.apache.org/proper/commons-crypto/

[32] M. Elsheimy, "C# Implementation of basic linear algebra concepts", https://github.com/elsheimy/Elsheimy.Components.Linears.

[33] Apache Commons Codec Library, https://commons.apache.org/proper/commons-codec/

[34] M. Bell, Michael, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*, Wiley & Sons, 2008.

[35] F. J. Massey, "The Kolmogrov-Smirnov test for goodness of fit", Journal of the American Statistical Association, vol. 46, no. 253, pp. 68–78, 1951.

[36] C. Withrow, "Error density and size in Ada software", IEEE Software, vol. 7, no. 1, pp. 26–30, 1990.

[37] V. R. Basili abd B. T. Perricone, "Software errors abd complexity: an empirical investigation", Communications of ACM, vol. 27, no. 1, pp. 42–52, 1984.

[38] Abramowitz and Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards and Applied Mathematics Series 55, 1970.

[39] D. J. Aigner, C. A. K. Lovell and P. Schmidt, "Formulation and estimation of stochastic frontier production functions", Journal of Econometrics, vol. 6, no. 1, pp. 21-–37, 1977.

[40] R. D. Banker and C. F. Kemerer, "Scale economies in new software development", IEEE Transactions on Software Engineering, vol. 10, no. 15, pp. 1199—1205, 1989.

[41] R. C. Martin, *Agile software development, principles, patterns and practices*, Prentice Hall, 2002.

[42] T. Gyimothy, R. Ferenc and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction", IEEE Transactions on Software Engineering, vol. 31, no. 10, pp. 897–910.

[43] P. Yu, T. Systa and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study", Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, pp. 99–107, 2002.

[44] K. S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*, Second edition, Wiley, 2008.

[45] S. Stevanetic and U. Zdun, "Exploring the relationships between the understandability of architectural components and graph-based component level metrics", Proceedings of 14th International Conference on Quality Software, pp. 353–358, 2014.

[46] W. Pan, H. Jiang, H. Ming, C. Chai, B. Chen and H. Li, "Characterizing software stability via change propagation simulation", Hindawi Complexity, pp. 1–17, 2019.

[47] Y. Li, Z. Wang, X. Zhong and F. Zou, "Identification of influential function modules within complex products and systems based on weighted and directed complex networks", Journal of Intelligent Manufacturing, vol. 30, no. 6, pp. 2375–2390, 2019.

[48] W. Whitt, "Untold horrors of the waiting room: What the equilibrium distribution will never tell about the queue-length process", Management Science, vol. 29, no. 4, pp. 395–408, 1983.

[49] H. A. Simon, "On a class of skew distribution functions", Biometrika, vol. 42, no. 3/4, pp. 425–440, 1955.

[50] G. Concas, M. Marchesi, S. Pinna and N. Serra, "Power-laws in a large object-oriented software system", IEEE Transactions on Software Engineering, vol. 33, no. 10, pp. 687–708, 2007.

[51] P. C. Pendharkar, J. A. Rodger and G. H. Subramanian, "An empirical study of the Cobb-Douglas production function properties of software development effort", Information and Software Technology, vol. 50, no. 12, pp. 1181–1188, 2008.

[52] P. C. Pendharkar, "Scale economies and production function estimation for object oriented software component and source code documentation size", European Journal of Operational Research, vol. 172, no. 3, pp. 1040–1050, 2006.

[53] D. R. Pai, G. H. Subramanian and P. C. Pendharkar, "Benchmarking software development productivity of CMMI Level 5 projects", Information Technology and Management, vol. 16, no. 3, pp. 235–251, 2015.

[54] M. Gell-mann and C. Tsallis, *Nonextensive Entropy: Interdisciplinary Applications*, Oxford University Press, 2004.

[55] Karmeshu and S. Sharma, "Power law and Tsallis entropy: network traffic and applications", in Chaos, Nonlinearity, Complexity, Springer, vol. 206, pp. 162–178, 2006.

[56] Karmeshu and S. Sharma, "Queue length distribution of network packet traffic: Tsallis entropy maximization with fractional moments", IEEE Communication Letters, vol. 10, no.1, pp. 34–36, 2006.

[57] Karmeshu and S. Sharma, "q-Exponential product-form solution of packet distribution in queueing networks: maximization of Tsallis entropy", IEEE Communication Letters, vol. 10, no. 8, pp.585 – 587, 2006.

[58] S. Sharma and Karmeshu, "Power law characteristics and loss probability: finite buffer queueing systems", IEEE Communication Letters, vol. 13, no. 12, pp. 971-9-73, 2009.