Process-Driven Incremental Effort Estimation

Kan Qi
University of Southern California
United States
kqi@usc.edu

Barry W. Boehm
University of Southern California
United States
boehm@usc.edu

***Abstract*—Effort estimation has shown its value in process decisions, such as feasibility analysis, resource allocation, risk mitigation, and project planning. In this paper, we propose an incremental effort estimation method that integrates phase-based effort estimation models to solve the concerns about software process compatibility and estimation accuracy, which one may often have when adopting effort estimation methods for project management.**

We define the process framework for incremental effort estimation in terms of the transition between the phase model that defines the analysis and design activities, the system models that specify a system's behavior and structure at different levels of detail, the sizing model that measures software functional size via transaction analysis, the Bayesian model that statistically models the effects that the size measurements have on project effort, and the phase-based effort estimation models that provide improved effort estimation accuracy over the targeted early phases.

The phase-based effort estimation models are calibrated and evaluated based on an empirical dataset of 61 master-level student projects from USC CSSE. The evaluation results show their improvements in out-of-sample estimation accuracy, provide a perspective about how estimation accuracy evolves throughout a software process, and set the practical criteria to decide the investment in analysis and design activities for the return in estimation accuracy.

***Index Terms*—Software process management and improvement, software functional size analysis, software size metrics, effort estimation, model calibration, Bayesian analysis**

I. Introduction

Effort estimation has shown its value in process decisions, such as feasibility analysis, resource allocation, risk mitigation, and project planning [1]. Software process compatibility and estimation accuracy are two major concerns one may often have when adopting an effort estimation practice [2] [3] [4] [5].

Software process compatibility is concerned in terms of what information can be derived from a software process for effort estimation and how the data conversion can be established. For example, existing effort estimation methods, such as the use case point and function point based models [6] [7] [8], can be applied to a software process by deriving the information, assumed by the software sizing models of the estimation models, from the system analysis and design results [9]. However, extra analysis effort and special task forces may be needed if the required information is not defined in the analysis and design activities of a software process [10]. Besides, since the analysis and design activities are usually exercised in a time order, which defines the logical phases of a software process, the existing estimation methods are only applicable at certain phases. Fig. 1, adopted from [11], illustrates the phase constraints when applying the existing size metrics to a software process. The constrained applicability of the existing methods prevents the practitioners from utilizing the updated knowledge gained over the process to achieve better accuracy, and may create obstacles to the agility of a software process.

Estimation accuracy directly influences the utility of the estimation results. Different software management decisions may require different degree of accuracy [2] [1]. For example, strategic decisions, such as feasibility analysis, have less requirement on accuracy than operational decisions, for example, risk mitigation and project planning and staffing [2]. Better estimation accuracy entails identifying more software size relevant information [12], which may require extra analysis effort if the analysis is not an integral part of the software process definition. Therefore, to achieve better software process efficiency, the balance between the investment in analysis effort and the return in estimation accuracy need to be considered when adopting an effort estimation method.

A solution to the aforesaid issues is to use a phase model that provides increasingly accurate software size information over the early phases for effort estimation. This paper identifies such a sequence of phases whose content provides increasingly accurate effort estimates. We summarize the advantages of the proposed incremental effort estimation method as follows:

1) Synchronizes the lifecycle of effort estimation with the lifecycle of a software project to allow the effort estimation practice to access the current analysis and design results and drive the current-stage project management decisions, which ensures software process compatibility.
2) Reuses the knowledge gained from the analysis and design activities at the earlier phases and incrementally integrates the later-phase information for effort estimation to boost the software process efficiency.
3) Keeps the effort estimation results updated with the best estimation accuracy possible to the knowledge at a certain stage.
4) Provides freedom of choice of conducting effort estimation at a specific phase or for a desired level of accuracy to improve the dynamics of a software process.

Our incremental effort estimation approach is defined into two strata: the process framework and the calibration method of the phase-based effort estimation models. The process framework is defined in terms of the transition between a series of typical software project models to lay the theoretical foundation. The models involved are: I. the phase model that defines the analysis and design activities adopted to specify the target system; II. the system models, as the results of the analysis and design activities, provide software size relevant information; III. the software sizing model that continuously measures software functional size by applying transaction analysis at the early phases; IV. the Bayesian model that statically models the posterior distributions of the effects that the size measurements have on project effort by considering both the prior beliefs as well as the sample information. V. the effort estimation models that provide phase-based effort estimates with improved accuracy. The calibration method provides the technical aspects of the process, which include the definition of the classification function, the Markov Chain Monte Carlo (MCMC) algorithm for calibrating the parameters, the iterative accuracy optimization process. With this structure, we aim to provide the adopters with two points of modification, such that the changes can either be made to the incremental effort estimation process to fit specific software processes and engineering environments, or the statistical analyses we propose to better calibrate the effort estimation models. Here we summarize the contributions of this paper:

1) Propose a process framework to lay the theoretical foundation for incremental software size analysis for project effort estimation.
2) Define specific size metrics that can be applied at early phases of a project, and the algorithm that calibrates effort estimation models defined using the metrics.
3) Provide three effective phase-based effort estimation models, which are calibrated and evaluated using an empirical dataset of 61 projects, and discuss the implications of the evaluation results on software process management.

This paper is structured as follows: in section II, we discuss the related work; section III introduces the process framework for incremental effort estimation; section IV presents the model calibration method and results; in section V, we evaluate the effort estimation models for their improvements in out-of-sample estimation accuracy and discuss their implications on software process management; section VI discusses the threats to validity; section VII concludes the research.

II. Related Work

To utilize the gradually better understanding about the personnel, platform, process, and product aspects during the progress of a project, COCOMO®II integrates two phase-based sub-models: the early-design and post-architecture models [2] [1] to provide early estimates for strategic decisions and later estimates with better estimation accuracy for operational decisions. Following the same idea of utilizing information available throughout the early phases for effort estimation, our approach focuses on modeling the phenomenon that the behavior and structure of the target system are specified in

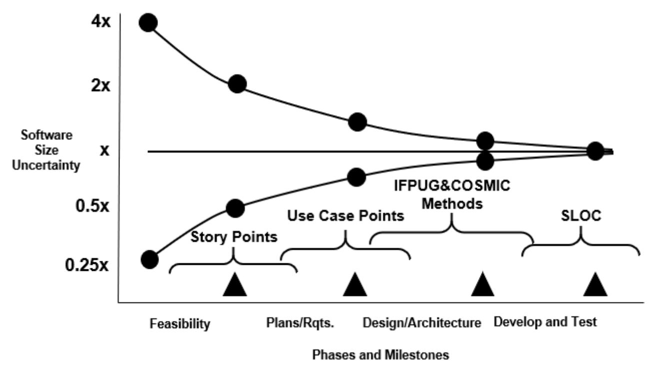

Fig. 1. Applicability of the Existing Size Metrics [11]

more detail over the time by the analysis and design activities. We derive the software functional size relevant information from the specifications for effort estimation. Therefore, our estimation method and evaluation results can complementarily serve COCOMO®II's objectives.

Many software functional size metrics have been proposed in the past for project effort estimation, among which IFPUG, COSMIC, and Mark-II function points are the most widely adopted ones [7] [13] [8] [14]. Those functional sizing methods model a software system in terms of transaction functions and data functions of different types. Transaction functions and data functions are identified as the elementary processes and logical data groups respectively. Their relative influences on software size are determined by the associated data elements. The software functional size is calculated as the sum of weighted transaction and data functions. Although our proposed size software sizing model shares the same functional form as the existing function point methods for the use of the sum of weighted transactions to measure software functional size, our method of weighting each transaction is fundamentally different. We identify three aspects of a transaction - the operational, structural, and data complexities to define the classification function that classifies the transactions into different complexity levels to distinguish their influences on project effort. This functional form serves our purpose of incremental effort estimation, which is a feature lacked in the existing software functional size metrics.

We employ a Bayesian model to statistically model the effects that transactions of different complexity levels have on project effort. Bayesian analysis has been similarly adopted by COCOMO®II to combine both domain experience and empirical study results about the effects that different cost drivers have on project effort [2] [15]. The use of Bayesian analysis corrected the unintuitive calibration results and improved effort estimation accuracy [15]. Different from the analytical approach used in COCOMO®II's calibration process, we use MCMC sampling method to simulate the full posterior distributions of the parameters, which allows the access of statistics such as means, variances, and confidence intervals to better assess the quality of the estimation models.

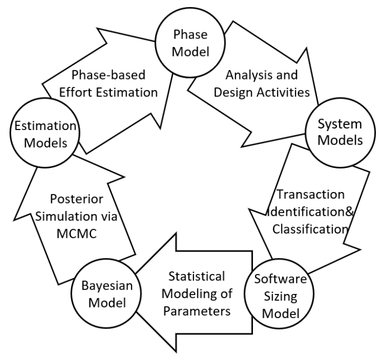
Phase Model
Analysis and Design Activities
System Models
Transaction Identification& Classification
Software Sizing Model
Statistical Modeling of Parameters
Bayesian Model
Posterior Simulation via MCMC
Estimation Models
Phase-based Effort Estimation

Fig. 2. The Underlying Models for Incremental Effort Estimation

III. MODEL DEFINITION

In this section, we define the process framework in terms of the transition between a series of software project models to lay the theoretical foundation of our incremental effort estimation method. The transition includes the information about what models we use, what processes we apply, and what models we derive for incremental effort analysis. The transition is depicted in Fig. 2. In the following sections, we provide brief discussions about the phase model and the software system models as they are summarized from the common methods, while elaborate on the definitions of the proposed transaction-based software sizing model, the incremental effort estimation models, and the Bayesian model.

A. The Phase Model

For incremental effort estimation, we tend to understand the derivation process of the system specifications that define the scope of a project at the early stage. The effective activities like requirements elicitation, system analysis, and architectural design are widely adopted by many successful software processes, for example, Use Case Driven Approach [16], ICSM [17], Resilient Agile [12], Rational Unified Process [18], etc. Some activities may be emphasized by certain processes while tailored by others to keep the balance between agility and rigorousness [19]. These activities are usually exercised in the time order, which define the logical phases of a project. Therefore, by abstraction, our targeted early phases for effort estimation are: requirement elicitation phase, system analysis phase, and architectural design phase. For the iterative agile or hybrid processes, those activities are distributed into iterations [17], therefore, the three phases can be regarded as the early phases of each iteration, and the effort estimation models defined based on the phases can be applied within the iterations.

B. The Software System Models

The results of the analysis and design activities are the system models that specify a software system at different levels of detail. For example, a system can be modeled by use cases as the interactions between the actors and a system by the requirement elicitation activity [16], as the components and their interactions by system behavior analysis for certain functional requirements [12], or in terms of methods and data elements defined in the platform specific architectural designs [20] [16] [17]. We utilize the information concerning the behavioral, structural, and data aspects of a software system, derived from the system models, to continuously analyze software functional size for incremental effort estimation.

C. The Transaction-based Sizing Model

To support incremental analysis of software functional size, we use "transaction" as the measurement unit, for its recurrence in the early-phase activities and its incrementally detailed definitions provided by different system models [16] [21]. The definition of transaction we adopt is given in *Definition 1*. The functional form of the software sizing model, presented in (1), is defined as the sum of weighted transactions.

Definition 1: (Transaction) A transaction is a sequence of operations exercised by system components to fulfill a type of interaction between an actor and a system.

$$Size = \sum_{t \in T} w_t \tag{1}$$

The weights reflect the effects that transactions of different complexity levels have on project effort. To determine the complexity level of a transaction, we tend to identify the factors that influence the effort for developing a transaction. For example, the number of operations that realize a transaction, the number of system components upon which the transaction is implemented, and the number of data element types that are associated with a transaction. Those attributes reflect the operational, structural, and data complexities of a transaction, which suggest the effort of understanding, implementing, and testing a transaction. The construction of the sizing model is supported by two transaction analytical operations - transaction identification and transaction classification.

1) Transaction Identification: Transaction identification involves the techniques of identifying transactions and their associated attributes from different analysis and design artifacts. Fig. 3 provides the graphic representation of the transaction model that we identify from sequence and class diagrams for software sizing using the method proposed in our previous research [9]. We use the following metrics to characterize the complexity of a transaction.

1) Transaction Length (TL), measures operational complexity of a transaction, by counting the number of the operations, identified at the system component level, which realize a transaction.
2) Transaction Degree (TD), measures the structural complexity of a transaction, using the average number of service methods of the components that implement a transaction.
3) Data Element Types (DETs), measures data complexity of transaction, by counting the number of data element types associated with a transaction.

To construct the transaction model, different types of UML diagrams can be employed. For example, a transaction can be identified as a set of steps from a use case description [22], as a set of activities from an activity diagram [20], and as an independent path from a robustness diagram [21], and as a

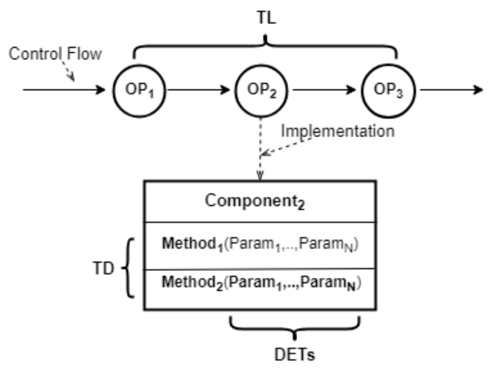

Fig. 3. The Transaction Model for Software Sizing

sequence of messages identified from a sequence diagram [9], to realize a type of user-system interaction. Those artifacts provide different levels of detailed description about a transaction, which determine the number of attributes we can use to define the classification function. Section III.D provides examples of identifying transactions and their associated attributes from different types of artifacts.

2) Transaction Classification: To classify transactions, we define the classification function based on the attributes identified from last step to map the multi-dimensional representation of a transaction into uni-dimensional representation in terms of the complexity levels. In section IV.B, we will introduce a classification function based on Manhattan distance to classify transactions into different complexity levels.

Transactions are differentiated by the complexity levels for their effects on development effort. Different weights are applied to model the effects in the size metrics proposed in section III.D. The weights can either be determined based on expert judgment, or statistically calibrated using empirical datasets. In section IV.C, we introduce a Bayesian approach to calibrate the weights in considering both experts' domain knowledge and sample information.

D. Incremental Effort Estimation Models

To incrementally estimate project effort, three software functional size metrics are defined to be countable at the defined three early phases. For each of the size metrics, we propose an effort estimation model to estimate project effort at the corresponding phase.

1) The Phase-based Software Size Analysis: Since different methods of identifying transactions provide different levels of detailed description about a transaction, which affect the number of attributes we can employ to classify the transactions. The proposed three size metrics employ the attributes of a transaction that can be measured at a certain phase. The size metrics share the same functional form as described in (1), while parameterized by different classification functions and weighting schemes. The size metrics are defined as follows:

$$\textit{SWT-I} = \sum_{t \in T} 1 \tag{2}$$

SWT-I (Sum of Weighted Transactions-I) adopts the simplest form of (1) by applying 1 to every transaction without differentiating the complexity of the transactions, which is simply the number of transactions. SWT-I is countable at the requirement elicitation phase by counting the number of user-system interactions from the use case narratives or the converted activity diagrams [22] [20].

$$\textit{SWT-II} = \sum_{t \in T} w^{*}(TD(t), TL(t)) \tag{3}$$

SWT-II can be applied at the analysis phase when the decomposition of system functionality into components and connectors is finished. TL and TD can be measured at this phase. For example, TL can be counted as the number of connectors that realize a transaction, and TD can be counted as the average number of inbound connectors of the components that implement a transaction. Our previous research [21] that identifies transactions as independent paths from robustness diagrams [16] can be used to derive the information. Component diagrams [23] can be similarly used by identifying the components and connectors that realize the transactions. The weight applied to a transaction is determined by TL and TD.

$$\textit{SWT-III} = \sum_{t \in T} w^{**}(TD(t), TL(t), DETs(t)) \tag{4}$$

At the design phase, with the detailed architecture design artifacts being available, for example, sequence and class diagrams, the connectors between the components are mapped into the methods with arguments and return values representing the referenced data elements. Based on this incremental information, we can further consider the influence from the data complexity on project effort. Specifically, SWT-III considers the number of data element types (DETs) associated with a transaction when applying the weights. Our previous research [9] provides an automated method of identifying TL, TD, and DETs of a transaction from sequence and class diagrams, which identifies a transaction as a sequence of messages that are further mapped into methods of classes in class diagrams. By this method, the sum of the number of argument types and the number of return value types for all the methods that implement a transaction is counted as DETs. TL and TD are similarly calculated as SWT-II.

2) The Parametric Effort Estimation Model: The effort estimation model models the effect that one unit of size measurement has on project effort, which can be generally defined as (5).

$$ProjectEffort = \alpha * SoftwareSize \tag{5}$$

Plugging the functional size metrics into (5), we derive the generic form of the incremental effort estimation model as (6).

$$Project\ Effort = \alpha * \sum_{t \in T} \sum_{i \in |W|} W_i * I_i(f_{cmplx}(t)) \tag{6}$$

Where, α is the effort adjustment factor,
T is the set of transactions,
W are the weights assigned to the $|W|$ complexity levels,
$I_i(x)$ is the indicator function,

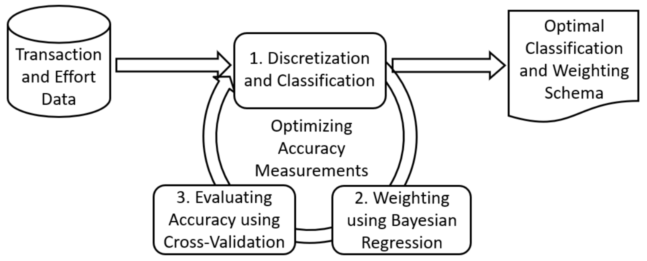
Transaction and Effort Data
1. Discretization and Classification
Optimal Classification and Weighting Schema
Optimizing Accuracy Measurements
3. Evaluating Accuracy using Cross-Validation
2. Weighting using Bayesian Regression

Fig. 4. The Model Calibration Process

$f_{cmplx}(t)$ classifies a transaction into a level of complexity. W and $f_{cmplx}(t)$ are specifically defined for different phases based on the number of transactional attributes measurable at a certain phase. The phase-based effort estimation models iteratively evaluate the transactional complexity over the early phases to provide incremental effort estimation.

E. The Bayesian Model

To calibrate the parameters of effort estimation models, we propose a Bayesian model to model the posterior probabilities of the parameters. Bayes' theorem described in (7) shows the process of updating the prior probability $P(A)$ of a random variable A with the likelihood $P(B|A)$ to generate the posterior probability $P(A|B)$ of A.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A) * P(A)}{P(B)} \tag{7}$$

Following the Bayes' theorem, we model the joint posterior probability $p(a, W|x, y)$ of the weights W and the effort adjustment factor α as proportional to the product of the likelihood $p(x, y|a, W)$ and the prior probabilities of the parameters: $p(a)$ and $p(W)$. The prior probabilities represent our prior beliefs about how the parameters affect project effort, while the likelihood represents how likely the observed effort data can be explained by the parametric effort estimation model defined in (6). The Bayesian model of the parameters is summarized in (8).

$$p(a, W|x, y) \propto p(x, y|a, W) * p(a) * p(W) \tag{8}$$

In section IV.C, we run Metropolis-Hastings MCMC algorithm to sample the joint posterior distribution of W and α based on the Bayesian model, and use the means of the marginal posterior distributions as the estimates of the parameters.

IV. Model Calibration

To calibrate the parameters of the proposed effort estimation models, we essentially answer the following two research questions:

1) what is the optimal classification of the transactions?
2) what weights should be assigned to the different complexity levels?

To answer the research questions, for each phase-based estimation model, we follow the iterative process depicted in Fig. 4, which has the following three major steps in each iteration:

1) Classify the transactions into different complexity levels by the classification function defined based on the discretized empirical distributions of the transactions over the considered dimensions.
2) Calibrate the weights applied to the different complexity levels using Bayesian analysis that updates our prior beliefs with the sample information derived from our empirical dataset.
3) Evaluate the out-of-sample effort estimation accuracy using the chosen estimation accuracy measure.

The three steps are iteratively executed to try different degree of discretization of the dimensions considered by a phase-based estimation model, and we use the classification function and weighting schema that provide the best accuracy measurement as the final estimates of its parameters. In the following sections, we elaborate the process through the case of the three dimensions - TL, TD, and DETs - used by SWT-III, while summarize the results for SWT-I and SWT-II.

A. The Data Set

The dataset we use to calibrate and evaluate the effort estimation models comprises 61 master's computer science student projects from USC's Center for Systems and Software Engineering (CSSE).

The projects lasted for 4-8 months and were developed during 2011-2018. The software products range from 1-10 KSLOC and are of different types: web applications, mobile applications, mobile games, information systems, and scientific tools, developed based on the requirements given by real-world clients from start-ups, non-profits, education institutes, government agencies, etc. The projects followed use case driven, plan driven, and agile methodologies with 5-8 members on the teams, who took specific roles, such as project manager, designer, architect, quality focal point, developer, and tester. The clients are closely involved to test and evaluate the products until their acceptance.

Project effort was recorded through Jira tickets and weekly effort reports, ranging from 100-3000 person hours. In total, 5797 transactions and their associated properties are assessed using the method introduced in [9].

B. Multi-dimensional Classification

To classify the transactions over the three dimensions - TL, TD, and DETs, we first discretize the distributions of the transactions over the individual dimensions using the quantile-based discretization strategy. This provides a set of cut points over the dimensions. Based on the cut points, a classification function is defined to classify the transactions into different complexity levels.

1) Quantile-based Discretization: To discretize the three dimensions, we first fit the gamma distribution function to the empirical frequency distribution for each of the three dimensions using maximum likelihood method, in order to find the distribution of the underlying population [24]. Gamma

TABLE I
GAMMA (Γ) FITTING AND CUT POINTS FOR TL, TD, AND DETs

	TL							TD							DETs						
Γ Fit	α: 6.54, β: 1.16, KS: p<0.01							α: 3.65, β: 0.70, KS: p<0.01							α: 1.66, β: 0.17, KS: p<0.01						
C.Pnts.	C_1^1	C_2^1	C_3^1	C_4^1	C_5^1	C_6^1	C_7^1	C_1^2	C_2^2	C_3^2	C_4^2	C_5^2	C_6^2	C_7^2	C_1^3	C_2^3	C_3^3	C_4^3	C_5^3	C_6^3	C_7^3
Cut_1	0.0	Inf	-/-	-/-	-/-	-/-	-/-	0.0	Inf	-/-	-/-	-/-	-/-	-/-	0.0	Inf	-/-	-/-	-/-	-/-	-/-
Cut_2	0.0	5.3	Inf	-/-	-/-	-/-	-/-	0.0	4.8	Inf	-/-	-/-	-/-	-/-	0.0	8.0	Inf	-/-	-/-	-/-	-/-
Cut_3	0.0	4.5	6.3	Inf	-/-	-/-	-/-	0.0	3.7	6.0	Inf	-/-	-/-	-/-	0.0	5.4	11.2	Inf	-/-	-/-	-/-
Cut_4	0.0	4.0	5.4	6.9	Inf	-/-	-/-	0.0	3.2	4.8	6.7	Inf	-/-	-/-	0.0	4.3	8.0	13.4	Inf	-/-	-/-
Cut_5	0.0	3.8	4.8	5.9	7.3	Inf	-/-	0.0	2.9	4.1	5.4	7.3	Inf	-/-	0.0	3.6	6.4	9.8	15.1	Inf	-/-
Cut_6	0.0	3.6	4.5	5.3	6.3	7.6	Inf	0.0	2.6	3.7	4.8	6.0	7.7	Inf	0.0	3.1	5.4	8.0	11.2	16.4	Inf

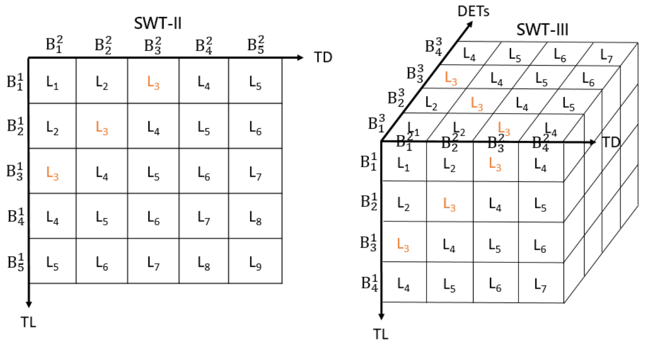
SWT-II
SWT-III
TD
TL
DETs
B_1^2 B_2^2 B_3^2 B_4^2 B_5^2
B_1^1 B_2^1 B_3^1 B_4^1 B_5^1
B_1^3 B_2^3 B_3^3 B_4^3
L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8 L_9

Fig. 5. Transaction Classification over Discretized Dimensions

distribution is chosen based on its goodness of fit to our empirical dataset introduced in section IV.A, in comparison with other typical right skewed distribution functions. In our experiment, we tried gamma distribution, log-normal distribution, log-logistic distribution, and Weibull distribution, and gamma distribution provides the best goodness of fit.

We also tested the significance of goodness of fit of gamma distributions using the bootstrap version of Kolmogorov-Smirnov (K-S) test [25], and the p-values being less than 0.01 suggest goodness of fit is significant. The shape parameter α and the rate parameter β of the fitted gamma distributions, and $p-$values from the K-S tests are provided in Table I.

We then discretized the gamma distribution functions by 1-6 quantities with 6 cutting operations Cut_1,...,Cut_6. Each cut operation Cut_t generates a set of cut points C_1^i,...,C_{t+1}^i for a dimension i ($i \in \{TL, TD, DETs\}$) to define t equal probability bins B_1^i,...,B_t^i where the transactions are categorized to generate a marginal discretized distribution. The cut points for the three dimensions, generated by the 6 cutting operations, are presented in Table I.

2) The Classification Function: The subscript (t) of each bin (B_t^i) represents the complexity level of a transaction with respect to the dimension i. The higher number the subscript is, the more complex the transaction is rated in that aspect. We then combine the bins of the three dimensions into a joint bin to classify the transactions over multiple dimensions. To combine the bins of the individual dimensions, we introduce the classification function $f_{complx}(t)$ defined based on the Manhattan distance between the origin and the coordinates of a transaction t, defined over the discretized dimensions. $f_{complx}(t)$ is defined in (9).

$$f_{cmplx}(t) = manhattan - distance_D(t) - |D| + 1$$
$$= \sum_{i<|D|} d_i(t) - |D| + 1 \quad (9)$$

Where, $D = \{d_1, d_2, ..., d_i\}$, represents the individual discretized dimensions. $|D|$ represents the number of dimensions that characterize a transaction. $d_i(t)$ returns the index of the bin into which a transaction t is categorized for dimension d_i. The Manhattan distance is shifted with a constant $|D| - 1$ to make the complexity levels starting from 1. This function maps measurements of complexity at the individual dimensions into a complexity level L_x. The larger Manhattan distance between a transaction and the origin, the higher complex level the transaction is rated at. This classification function also provides max number of the complexity levels, which can be calculated by (10).

$$max(f_{cmplx}) = \sum_{i\in|D|} |d_i| - |D| + 1 \quad (10)$$

Where, $|d_i|$ is the number of bins for dimension d_i.

In our case, we use the TL, TD, and DETs dimensions to characterize the transactions, and the measurements for the three dimensions are used to determine the complexity level of a transaction. For example, under discretization of 4 bins for the three dimensions, the max number of complexity levels $max(f_{cmplx})$ can be calculated by (11).

$$max(f_{cmplx}) = 4 + 4 + 4 - 3 + 1 = 10 \quad (11)$$

A transaction t that is rated as 6, 7, and 4 for TL, TD, and DETs respectively, has coordinates of $\{3, 4, 1\}$ over the discretized dimensions according to Table I and can be classified into the complexity level of 6 by Eq. (12).

$$f_{cmplx_{10}}(t) = 3 + 4 + 1 - 3 + 1 = 6 \quad (12)$$

Exemplary frequency distributions of the classified transactions are provided in Fig. 6, which are derived by classifying the transactions in our empirical dataset with 1-6 bins of discretization of the TL and TD dimensions. The classified transaction distributions over the three dimensions are skipped to save the space. As we can see from the Fig. 6, the

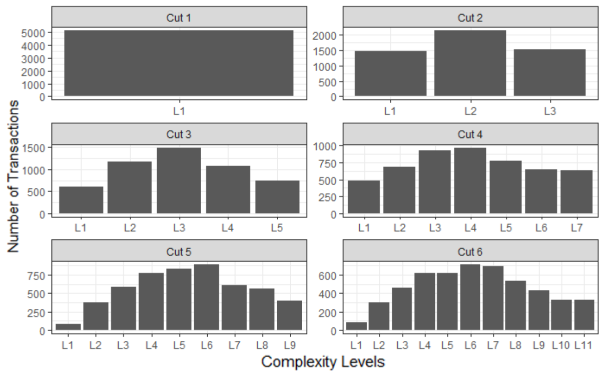

Fig. 6. Transaction Distributions over Complexity Levels

distributions are well balanced.

C. Weighting using Bayesian Analysis

The weights W assigned to the complexity levels and the effort adjustment factor α are statistically modeled by (8), introduced in section III.E. In the following sections, we will introduce the definitions of the major components of the Bayesian model and our approach of estimating the parameters using Metropolis-Hastings MCMC algorithm.

1) The Priors: The Bayesian model employs the priors on the expected values of the weights W assigned to the complexity levels, the covariance matrix Σ of W, the effort adjustment factor α, and the standard deviation σ of the residuals. Based on the widely observed law "diseconomy of scale" in project effort estimation studies [1] [26], we model the hypothetical non-linearly increasing effects that different complexity levels have on project effort using Fibonacci sequence, which is, the prior expected values of weights $\hat{W}$ are assigned with numbers from Fibonacci sequence. Fibonacci sequence is similarly used by Story Points [26] to decide the relative efforts for the tasks of different complexity levels. An example of the Fibonacci sequence numbers we use is as follows:

$$1, 2, 3, 5, 8, 13, ... \tag{13}$$

The variances of the weights Δ are set to $(1/3 * (w_i - w_{i-1}))^2$ to avoid the potential overlaps in the final calibration results of the parameters. The weights are modeled as independent to each other, therefore the prior covariance matrix $\hat{\Sigma}$ is diagonalized with Δ.

For the priors on α and σ, we use non-informative priors such that the parameters would be largely determined by the sample information [27]. For instance, we use uniform distribution ($unif(0, 30)$) for α and Jeffreys Prior ($1/\sigma$) for σ [27].

2) The likelihood function: To derive the likelihood of the parameters, we run multiple linear regression analysis of (14) on our empirical dataset, which is the expanded form of (6).

$$Effort = \alpha * (w_1 T_{L_1} + w_2 T_{L_2} + w_3 T_{L_3} + ... + w_n T_{L_n}) \tag{14}$$

Algorithm 1 Posterior Estimation of W and α

```
Data: $\{y_i, x_i\}_{i=1}^n$ - regression data;
$\hat{W}$ - prior means of weights;
$\hat{\Sigma}$ - prior covariance matrix of weights;
Result: The posterior estimates of the weights $W^*$ and the effort adjustment factor $\alpha^*$;
1. Initialize:
2. pick an initial state $\epsilon_0 = \{\alpha_0, W_0, \sigma_0\}$; set $t = 0$;
3. Iterate: t $< 100000$
4. Propose: randomly generate a candidate state $\epsilon' = \{\alpha', W', \sigma'\}$, such that: $\alpha' \sim norm(\alpha_t, c_1)$, $W' \sim norm(W_t, c_2)$, $\sigma' \sim norm(\sigma_t, c_3)$;
5. Calculate the acceptance probability:
6. $A(\epsilon', \epsilon_t) = posterior(\epsilon'_t)/posterior(\epsilon_t)$,
7. where $posterior(\epsilon) = prior(\epsilon) * likelihood(\epsilon)$;
8. $prior(\epsilon) = punif(\epsilon.\alpha, 0, 15) * pnorm(\epsilon.W, \hat{W}, \hat{\Sigma}) * (1/\epsilon.\sigma)$;
9. $likelihood(\epsilon) = \prod_{i=1}^n pnorm(y_i, \epsilon.\alpha * \epsilon.W * x_i, \epsilon.\sigma)$.
10. Accept or Reject:
11. generate a uniform random number $u \in [0, 1]$;
12. if $u \leq A(\epsilon', \epsilon_t)$, set $\epsilon_{t+1} = \epsilon'$;
13. if $u > A(\epsilon', \epsilon_t)$, set $\epsilon_{t+1} = \epsilon_t$;
14. Increment: set $t = t + 1$;
15. Output:
16. remove the first 5000 iterations as burnin;
17. $\{W^*, \alpha^*\}$ = means($\epsilon.W$, $\epsilon.\alpha$);
```

where, $W = \{w_1, w_2, ..., w_n\}$ represents the weights that should be assigned to the complexity levels $L_1, L_2, ..., L_n$, and α represents the effort adjustment factor.

The likelihood of the parameters W and α is calculated as the probability of the residuals under the normal distribution $norm(0, \sigma)$. The residuals are calculated as the differences between the actual effort and the effort estimated by (14). Therefore, based on our empirical dataset, the likelihood of the parameters can be calculated.

3) Simulating Posterior Distributions Using MCMC: Based on the Bayesian model, we run Metropolis-Hastings (H-M) algorithm to simulate the joint posterior probability distribution of α and W. The algorithm comprises three major steps: I. it takes a random starting point ϵ_0 from the parameter space as the current state ϵ_t; II. it iteratively takes random moves ϵ', generated by a proposal function, around ϵ_t in the parameter space; III. If the posterior probability $posterior(\epsilon')$ evaluated at ϵ' is higher than ϵ_t's posterior probability $posterior(\epsilon_t)$, ϵ' would be accepted as the next state ϵ_{t+1}. These three steps are iterated by a fixed number of times to generate a chain of random states. M-H algorithm ensures that this chain of sampled states ϵ converges to the posterior probability distribution of the parameters [28].

Algorithm 1 summarizes our approach of generating posterior distributions using M-H algorithm. We use Gaussian density functions in the proposal function (line 4 of Algorithm 1). For example, c_1, c_2, and c_3 are standard deviations of the normal distributions used to generate random candidate states, which, in our case, are set 0.1, $1/5 * (w_i - w_{i-1})$, and 3

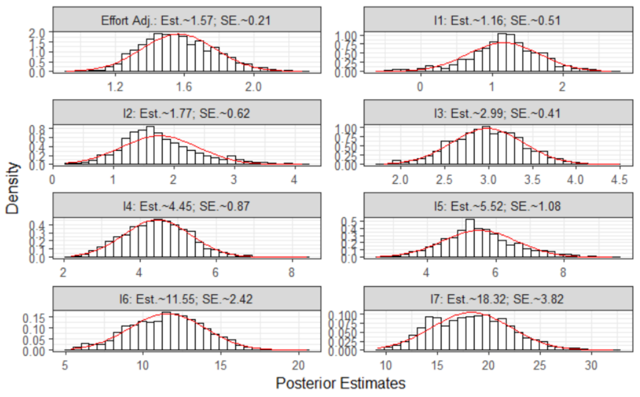

Fig. 7. Posterior Distributions of the Parameters

respectively. Those are the hyper parameters of the algorithm, which can be locally tuned for the best performance. To ensure the chain of sampled states converges to the actual posterior distribution, we run 100000 iterations with the first 5000 sampled states removed as the burn-in, and observe the acceptance rate to make sure it falls into the reasonable range of 20%-50% as suggested in [28]. The means of the sampled marginal distributions are used as the estimates of the parameters, and the standard errors are calculated to assess the variability of the parameters.

An example of the marginal posterior distributions of the parameters are given in Fig. 7, in which the transactions are classified into 7 complexity levels using (9) under the discretization of 3 bins for the three dimensions using the cut points provided in Table I. Correspondingly, 7 marginal posterior distributions of the weights are sampled using M-H algorithm. The means and the standard errors of the posterior distributions are calculated and annotated in Fig. 7.

D. Iterative Evaluation of Estimation Accuracy

To find the optimal classification and weighting schema, we evaluate the out-of-sample estimation accuracy in terms of PRED(.25) under different ways of classification, defined by 1-6 bins of discretization of the dimensions employed by the effort estimation models. PRED(.25) is used for its superior reliability in selecting the true model and robustness to outliers compared with the other typically used accuracy measure MMRE [29]. As defined in (15), PRED(x) measures the percentage of the estimates within a threshold x in terms of MRE. High values of PRED(x) are desirable. This statistic shows how often estimates can be expected to be within an acceptable margin of error.

$$PRED(x) = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1, & \text{if } MRE_i \leq x, where\ MRE_i = \frac{|y_i - \hat{y}|}{y_i} \\ 0, & \text{otherwise}\end{cases} \tag{15}$$

In each iteration of classification, the empirical dataset is separated into 10 folds to exercise the cross-validation process in 10 runs. The cross-validation process tests the ability of the trained models in predicting new cases [24]. Specifically, in each run of the cross-validation, we use 9 folds as the training set to exercise the steps introduced in section IV.B and section IV.C to calibrate the parameters, and use 1 fold to evaluate effort estimation accuracy of the calibrated model using PRED(.25). We use the average of the PRED(.25) results of the 10 runs as the final measurement of accuracy.

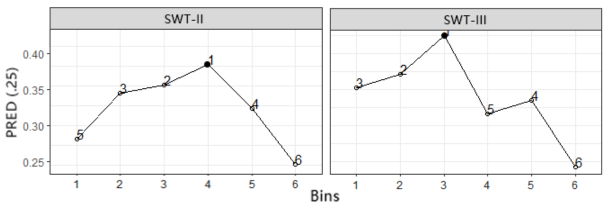

Fig. 8. Evaluating PRED(.25) under 1-6 Bins of Discretization

For 6 iterations of classification, 6 effort estimation models are defined, calibrated, and evaluated. We rank the models in terms of their PRED(.25) measurements to indicate their performance. The parameters under the binning that provides the highest PRED(.25) measurement are used as the final set of values that define a phased-based effort estimation model.

E. The Calibration Results

The accuracy measurements for SWT-II and SWT-III calculated over 1-6 binning of the considered dimensions are presented in Fig. 8. Since SWT-I only uses the number of transactions as the size metric, all the transactions are classified into one complexity level and no iteration is applied. We can see that the best binning for SWT-II is 4 and the best binning for SWT-III is 3, since they provide the highest PRED(.25) values in comparison with other models.

The calibration results for the parameters of the phase-based effort estimation models are presented in Table II. For SWT-I, the classification function $f^1_{cmplx_1}(t)$ only returns one complexity level, and the weight w_1 for the complexity level and the effort adjustment factor α are calibrated through Bayesian analysis. For SWT-II, since the best binning is 4, we have total 7 complexity levels. The two sets of cut points C_1,...,C_5, the classification function $f^2_{cmplx_7}(t)$, the weights w_1,...,w_7, and the effort adjustment factor α are provided to define the effort estimation model. For SWT-III, similarly we have $f^3_{cmplx_7}(t)$ defined using the 3 sets of cut points C_1,...,C_4 generated by 3 binning of the three dimensions. The weights w_1,...,w_7 and the effort adjustment factor α are also provided. Based on the simulated posterior distributions, we provide the standard errors (SE.) and 95% confidence intervals (CI.L$\sim$CI.U) of the parameters in Table II for the model adopters to construct interval estimates of project effort [1].

V. MODEL EVALUATION

In this section, we present the accuracy evaluation results for the proposed effort estimation models and discuss their implications on software process management.

A. Comparison with IFPUG Function Points

To show the effectiveness of the proposed effort estimation models, we compare them with a baseline effort estimation model defined based on the widely used function point size metric IFPUG [7], which is given in (16).

TABLE II
THE CALIBRATED MODEL PARAMETERS

Parameters	SWT-I				SWT-II					SWT-III			
Adj. Fctr.	Est.	SE.	CI.L	CI.U	Est.	SE.	CI.L	CI.U		Est.	SE.	CI.L	CI.U
α	4.87	1.12	3.15	7.16	0.80	0.13	0.55	1.09		1.57	0.21	1.21	2.02
Cut Pnts.	C				C_1	C_2	C_3	C_4	C_5	C_1	C_2	C_3	C_4
TL	-/-				0	4.0	5.4	6.9	Inf	0	4.5	6.3	Inf
TD	-/-				0	3.2	4.8	6.7	Inf	0	3.7	6.0	Inf
DETs	-/-				-/-	-/-	-/-	-/-	-/-	0	5.4	11.2	Inf
Cls. Func.	$f^1_{cmplx_1}(t)$				$f^2_{cmplx_7}(t)$					$f^3_{cmplx_7}(t)$			
	1				$\sum_{d\in\{TL,TD\}} d(t) - 1$					$\sum_{d\in\{TL,TD,DETs\}} d(t) - 2$			
Weights	Est.	SE.	CI.L	CI.U	Est.	SE.	CI.L	CI.U		Est.	SE.	CI.L	CI.U
w_1	1.28	0.30	0.81	1.91	1.20	0.70	0.35	2.48		1.16	0.51	0.17	2.11
w_2	-/-	-/-	-/-	-/-	1.77	0.68	0.45	3.16		1.77	0.62	0.74	3.24
w_3	-/-	-/-	-/-	-/-	2.88	0.44	2.01	3.75		2.99	0.41	2.19	3.80
w_4	-/-	-/-	-/-	-/-	4.42	0.87	2.78	6.17		4.45	0.87	2.79	6.21
w_5	-/-	-/-	-/-	-/-	7.15	1.42	4.56	9.97		5.52	1.08	3.57	7.98
w_6	-/-	-/-	-/-	-/-	11.87	2.27	7.86	16.43		11.55	2.42	6.59	15.97
w_7	-/-	-/-	-/-	-/-	13.19	2.82	8.71	19.76		18.32	3.82	11.85	26.15

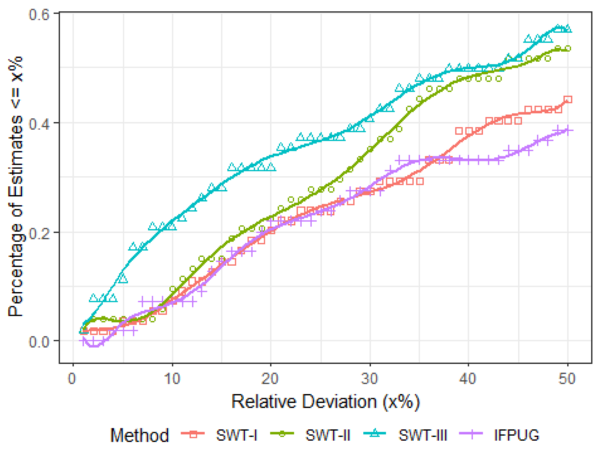

Fig. 9. Comparison over PRED(.01) - PRED(.50)

$$Effort = \alpha * IFPUG \tag{16}$$

The IFPUG function point measurements are derived from the sequence and class diagrams of the 61 projects using the method introduced in [30]. We evaluate the out-of-sample accuracy of the three proposed models and the IFPUG model using the similar 10-fold cross-validation process introduced in section IV.D. The difference is that we evaluate the out-of-sample accuracy from PRED(.01) to PRED(.50) for the four models. The evaluation results are presented in Fig. 9, which provides a clear view about the performance of the models. As we can observe from Fig. 9, SWT-II and SWT-III provide noticeable improvements over the IFPUG model, while SWT-I provides comparable performance as IFPUG. Therefore, the calibrated models can be used to predict project effort with satisfactory estimation accuracy.

B. Relative Improvements in Estimation Accuracy

Based on the accuracy evaluation results presented in Fig. 9, we further compare the relative improvements in accuracy for the proposed effort estimation models. We take three points from Fig. 9, and present them in Table III. As we can see, SWT-III provides the best performance, since it provides the highest values for both PRED(.15), PRED(.25), and PRED(.50). SWT-II also provides certain degree of improvement over SWT-I for the three accuracy measures. We still use PRED(.25) to make the conclusion about their relative improvements in estimation accuracy for the reasons mentioned in Section IV.D. Therefore, under PRED(.25), SWT-III performs better than SWT-II by 9.3%, while SWT-II performs better than SWT-I by 4.0%. We conclude that the inclusion of data complexity in transaction classification significantly improves estimation accuracy and the later-phase models provide better effort estimation accuracy than the earlier-phase models.

TABLE III
PRED(.15), PRED(.25), PRED(.50) FOR THE COMPARATIVE MODELS

Model	PRED(.15)	PRED(.25)	PRED(.50)
SWT-I	14.5%	23.8%	46.2%
SWT-II	15.1%	27.8%	53.5%
SWT-III	28.0%	37.1%	57.1%
IFPUG	14.7%	23.8%	38.5%

C. Implications on Software Process Management

The implications of the accuracy evaluation results on software process management are two-fold. First, it provides a clear understanding about how much accuracy can be improved by integrating information available in the later phases. If a certain level of estimation accuracy is desired for certain project management decisions, project managers can choose to integrate the corresponding phase(s) of effort estimation into their current system analysis and design practice. Second, if accuracy level is not mandatory, one may need to balance between the investment of analysis effort and the return in estimation accuracy when they make the decision about what analysis to adopt for effort estimation. The evaluation results provide the criteria for such a trade-off analysis. For example, as Jacobson mentioned in [16], the effort required for designing implementation environment specific classes is commonly 5-10 times more than the effort required for the classes derived through domain object analysis. Adopting the sequence and class diagram based design activities, or other design activities at similar levels of detailedness, may not be worth the investment, if the goal is only to provide data for effort estimation, since the return on investment ratio of doing such design activities is merely 13%-27% of doing domain object analysis, in considering SWT-III is relatively about 33% better than SWT-II.

VI. Threats to Validity

The main threat to validity of our proposed method and the experimental results is the potential bias that exists in the dataset used for model calibration and evaluation, which may come from the following two aspects. First, although the proposed transaction analysis is intended for different types of analysis and design artifacts, we only used the method described in [9] to identify transactions from sequence and class diagrams. The parameters may need to be updated for other types of artifacts. Second, the studied projects are considered small projects for the delivered software products range from 1-10 KSLOC and are developed by 5-8 team members. Therefore, the accuracy evaluation results presented in this paper may not be directly applicable to larger projects. Those weaknesses can be mitigated by collecting more data points from a wider range of software projects and more types of analysis and design artifacts, so that we can re-calibrate the models and evaluate the statistical significance of the accuracy improvements.

VII. Conclusions

To summarize, we provide a process framework and an incremental effort estimation method that provides phase-based effort estimates by continuously integrating software functional size relevant information over the early phases of a software project. The process framework is defined in terms of the transition of a series of software project models. Based on the framework, three phase-based effort estimation models are defined and calibrated through Bayesian analysis of an empirical dataset of 61 master's student team projects. We evaluate the out-of-sample accuracy of the three models using 10-fold cross-validation, and show that the models provide satisfactory estimation accuracy and can facilitate the managerial decision of selecting the estimation models that meet the expectations of estimation accuracy or the return on the investment in analysis effort.

The future work includes integrating more transaction identification techniques into the framework to identify transactions from more types of analysis and design artifacts and software projects, such that we can reduce the the bias discussed in section VI. The other side of the trade-off analysis is the quantitative results for the efforts of doing different kinds of analysis and design activities, with which we are able to provide a precise equation for the trade-off analysis.

References

[1] Barry W Boehm. Software engineering economics. In *Software pioneers*, pages 641–686. Springer, 2002.

[2] Barry W. Boehm. *Software cost estimation with Cocomo II*. Prentice Hall, Upper Saddle River, NJ, 2000.

[3] B. Anda, E. Angelvik, and K. Ribu. Improving estimation practices by applying use case models. In *PROFES*, pages 383–397. Springer, 2002.

[4] Steven D Sheetz, David Henderson, and Linda Wallace. Understanding developer and manager perceptions of function points and source lines of code. *Journal of Systems and Software*, 82(9):1540–1549, 2009.

[5] Bruce Lo and Xiangzhu Gao. Assessing software cost estimation models: criteria for accuracy, consistency and regression. *Australasian Journal of Information Systems*, 5(1), 1997.

[6] Gustav Karner. Resource estimation for objectory projects. *Objective Systems SF AB*, 17, 1993.

[7] A.J. Albrecht. Function point analysis. In *Encyclopedia of Software Engineering*, volume 1, pages 518–524. Wiley, 1994.

[8] Charles R Symons. *Software sizing and estimating: Mk II FPA (function point analysis)*. John Wiley & Sons, Inc., 1991.

[9] Kan Qi and Barry W Boehm. Detailed use case points (ducps): a size metric automatically countable from sequence and class diagrams. In *Proceedings of the 10th International Workshop on Modelling in Software Engineering*, pages 17–24. ACM, 2018.

[10] Silvia Abrahao and Oscar Pastor. Measuring the functional size of web applications. *International Journal of Web Engineering and Technology*, 1(1):5–16, 2003.

[11] Anandi Hira. Alternative size metrics for software estimation. *USC CSSE*, 2017.

[12] Doug Rosenberg, Barry Boehm, Bo Wang, and Kan Qi. Rapid, evolutionary, reliable, scalable system and software development: The resilient agile process. In *Proceedings of the 2017 International Conference on Software and System Process*, pages 60–69. ACM, 2017.

[13] S Oligny, A Abran, and C Symons. Cosmic-ffp some results from the field trials. In *15th International Forum on COCOMO and Software Cost Estimation*, 2000.

[14] Anandi Hira and B. Boehm. Function point analysis for software maintenance. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016.

[15] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transactions on Software Engineering*, 25(4):573–583, 1999.

[16] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach*. ACM Press;Addison-Wesley Pub, 1992.

[17] Barry Boehm and Richard Turner. The incremental commitment spiral model (icsm): principles and practices for successful systems and software. In *Proceedings of the 2015 International Conference on Software and System Process*, pages 175–176. ACM, 2015.

[18] Philippe Kruchten. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[19] Barry Boehm and Richard Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.

[20] Matt Stephens and Doug Rosenberg. *Design driven testing: test smarter, not harder*. Apress, 2011.

[21] Kan Qi and Barry W Boehm. A light-weight incremental effort estimation model for use case driven projects. In *Software Technology Conference (STC), 2017 IEEE 28th Annual*, pages 1–8. IEEE, 2017.

[22] R. Collaris and E. Dekker. Software cost estimation using use case points: Getting use case transactions straight. *IBM Developer*, 2009.

[23] D. Bell. Uml basics: The component diagram. *IBM Global Services*, 2004.

[24] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York, 2001.

[25] J. Praestgaard. Permutation and bootstrap kolmogorov-smirnov tests for the equality of two distributions. *Scandinavian Journal of Statistics*, 1995.

[26] Evita Coelho and Anirban Basu. Effort estimation in agile software development using story points. *International Journal of Applied Information Systems (IJAIS)*, 3(7), 2012.

[27] Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proc. R. Soc. Lond. A*, 186(1007):453–461, 1946.

[28] G. O Roberts, A. Gelman, and W. R Gilks. Weak convergence and optimal scaling of random walk metropolis algorithms. *The annals of applied probability*, 1997.

[29] D. Port, V. Nguyen, and T. Menzies. Studies of confidence in software cost estimation research based on the criterions mmre and pred, 2009.

[30] T. Uemura, S. Kusumoto, and K. Inoue. Function point measurement tool for uml design specification. In *Software Metrics Symposium, 1999. Proceedings. Sixth International*. IEEE.