The Use of Simulation Techniques for Hybrid Software Cost Estimation and Risk Analysis

Michael Kläs, Adam Trendowicz, Axel Wickenkamp, Jürgen Münch

Fraunhofer Institute for Experimental Software Engineering

Fraunhofer-Platz 1
67663 Kaiserslautern, Germany

+49-631-6800-0

{michael.klaes; adam.trendowicz; axel.wickenkamp; juergen.muench} @iese.fraunhofer.de

Nahomi Kikuchi

Oki Electric Industry Co., Ltd.

1-16-8 Chuou, Warabi-shi, Saitama 335-8510, Japan

+81-48-431-7211

kikuchi386@oki.com

Yasushi Ishigai

Information-technology Promotion Agency Software Engineering Center

2-28-8 Honkomagome, Bunkyo-Ku, Tokyo, 113-6591, Japan

+81-3-5978-7543

ishigai@ipa.go.jp

ABSTRACT

Cost estimation is a crucial field for companies developing software or software-intensive systems. Besides point estimates, effective project management also requires information about cost-related project risks, e.g., a probability distribution of project costs. One possibility to provide such information is the application of Monte Carlo simulation. However, it is not clear whether other simulation techniques exist that are more accurate or efficient when applied in this context. We investigate this question with CoBRA®[1], a cost estimation method that applies simulation, i.e., random sampling, for cost estimation. This chapter presents an empirical study, which evaluates selected sampling techniques employed within the CoBRA® method. One result of this study is that the usage of Latin Hypercube sampling can improve average simulation accuracy by 60% and efficiency by 77%. Moreover, analytical solutions are compared with sampling methods, and related work, limitations of the study, and future research directions are described. In addition, the chapter presents a comprehensive overview and comparison of existing software effort estimation methods.

Keywords

Cost estimation, CoBRA®, simulation technique, empirical evaluation.

TABLE OF CONTENT

[1] CoBRA is a registered trademark of the Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany.

1. INTRODUCTION

Rapid growth in the demand for high quality software and increased investment into software projects show that software development is one of the key markets worldwide [28], [29]. A fast changing market demands software products with ever more functionality, higher reliability, and higher performance. Moreover, in order to stay competitive, software providers must ensure that software products are delivered on time, within budget, and to an agreed level of quality, or even with reduced development costs and time. This illustrates the necessity for reliable software cost estimation, since many software organizations budget unrealistic software costs, work within tight schedules, and finish their projects behind schedule and budget, or do not complete them at all [101].

At the same time, software cost estimation is considered to be more difficult than cost estimation in other industries. This is mainly because software organizations typically develop products as opposed to fabricating the same product over and over again. Moreover, software development is a human-based activity with extreme uncertainties. This leads to many difficulties in cost estimation, especially in early project phases. These difficulties are related to a variety of practical issues. Examples include difficulties with project sizing, a large number of associated and unknown cost factors, applicability of cost models across different organizational contexts, or, finally, insufficient data to build a reliable estimation model on. To address these and many other issues, considerable research has been directed at gaining a better understanding of the software development processes, and at building and evaluating software cost estimation techniques, methods, and tools [12].

Traditionally, effort estimation has been used for the purpose of planning and tracking project resources. Effort estimation methods that grew upon those objectives do not, however, support systematic and reliable analysis of the causal effort dependencies when projects fail. Currently software industry requires effort estimation methods to support them in understanding their business and identifying potential sources of short-term project risks and areas of long-term process improvements. Moreover, in order to gain wider industrial acceptance, a candidate method should minimize the required overhead, e.g., by utilizing a variety of already existing information sources instead of requiring extensive expert involvement and/or large project measurement databases. Yet, existing estimation methods (especially those currently used in the software industry) do not offer such comprehensive support.

An example prerequisite to accepting a certain estimation method is its applicability within a particular context, which includes its adaptability to organization-specific characteristics such as availability of required data or effort required to apply the method. Usually, the latter two issues are contradictory: the less effort a method requires to build the estimation model, the more measurement data from previous projects is needed[2]. Data-based methods focus on the latter exclusively. Contrariwise, expert-based methods require almost no measurement data, but obtaining estimates costs a lot of effort. Software organizations move between those two extremes, tempted either by low application costs or low data re-

[2] The more data is available, the less expert involvement is required and the company's effort is reduced. This does not include the effort spent on collecting the data. Above a certain maturity level, companies have to collect the data needed anyway.

quirements. In fact, a great majority of organizations that actually use data-based methods do not have a sufficient amount of appropriate (valid, homogeneous, etc.) data as required by such methods. Hybrid methods offer a reasonable bias between data and effort requirements, providing at the same time reliable estimates and justifiable effort to apply the method [14], [18], [70].

Moreover, estimation methods are required to cope with the uncertainty inherent to software development itself as well as to the estimation activity. On the one hand, the preferred method should accept uncertain inputs. On the other hand, besides simple point estimates means to draw conclusions about the estimation uncertainty and effort related-project risk should be provided. The use of probabilistic simulation provides the possibility to deal with estimation uncertainty, perform cost risk analyses, and provide an add-on of important information for project planning (e.g., how probable it is not to exceed a given budget).

One of the methods that respond to current industrial objectives with respect to effort estimation is CoBRA® [14], an estimation method developed at the Fraunhofer Institute for Experimental Software Engineering (IESE). The method uses sparse measurement data (size, effort) from already completed software projects in order to model development productivity and complements it with expert evaluations in order to explicitly model the influence of various project characteristics on productivity deviations.

Random simulation is one of the core elements of the CoBRA® method; it is supposed to deal with estimation uncertainty introduced through expert evaluations. Experts first specify a causal model that describes which factors influence costs (and each other) and how. Afterwards, they quantify the impact of each factor on costs by giving the maximal, minimal, and most likely increase of costs dependent on a certain factor. The simulation component of CoBRA® processes this information into a probability distribution of project costs. This provides decision makers with a robust basis for managing software development costs, e.g., planning software costs, analyzing and mitigating cost-related software risks, or benchmarking projects with respect to software costs. Numerous practical benefits of CoBRA® have been proven in various industrial applications (e.g., [14], [98]).

Originally, CoBRA® utilized the Monte Carlo (MC) simulation technique. Yet, experience from applying the CoBRA® method in an industrial project [14] suggested that Monte Carlo might not be the optimal solution for that purpose. Latin Hypercube (LH) was proposed as an alternative method that can improve the performance of the CoBRA® implementation. Yet, neither a definition of the performance nor empirical evidence is available that would support this hypothesis. This led us to the question of whether there exist techniques that deliver more accurate results in the context of software cost estimation in general, and for the CoBRA® method in particular, than simple Monte Carlo sampling or performing the simulation in a more efficient way. In addition, we are interested in the extent of accuracy improvement that can be expected from the use of such a technique.

In this chapter, we describe analytical considerations as well as the results of an empirical study we conducted in order to answer these questions. Our major objective was to evaluate the magnitude of possible CoBRA® accuracy and efficiency improvement related to the selection of a certain simulation technique and its parameters. In order to achieve our goal, we derived several analytical error estimations and compared selected simulation techniques (including MC and LH) in an experiment employing various settings (i.e., parameters) within the same CoBRA® application (i.e., on the same input data).

The chapter is organized as follows: Section 2 presents the necessary theoretical foundations regarding the CoBRA® cost estimation method and the simulation techniques relevant for this chapter. Section 3 provides an overview of related work. It presents a comprehensive overview and comparison of existing

estimation methods as a well as summary of common simulation techniques. In Section 4, the research questions are described. Section 5 presents analytical results regarding one of the research questions. Section 6 sketches simulation algorithms that were selected for comparison and motivates why an empirical study is necessary for such a comparison. Section 7 presents the empirical study, including its results and limitations. Finally, Section 8 summarizes the findings of the study and outlines perspectives of further research work.

2. BACKGROUND

2.1 CoBRA® Principles

CoBRA® is a hybrid method combining data- and expert-based cost estimation approaches [14]. The CoBRA® method is based on the idea that project costs consist of two basic components: nominal project costs (Equation 2.1.1) and a cost overhead portion (Equation 2.1.2).

Equation 2.1.1

$$Cost = \underbrace{Nominal\ Productivity \cdot Size}_{Nominal\ Cost} + Cost\ Overhead$$

Equation 2.1.2

$$Cost\ Overhead = \sum_i Multiplier_i(Cost\ Factor_i) + \sum_i \sum_j Multiplier_{ij}(Cost\ Factor_i, Indirect\ Cost\ Factor_j)$$

Nominal cost is the cost spent only on developing a software product of a certain size in the context of a *nominal* project. A nominal project is a hypothetical "ideal" project in a certain environment of an organization (or business unit). It is a project that runs under optimal conditions; i.e., all project characteristics are the best possible ones ("perfect") at the start of the project. For instance, the project objectives are well defined and understood by all staff members and the customer and all key people in the project have appropriate skills to successfully conduct the project. Cost overhead is the additional cost spent on overcoming imperfections of a real project environment such as insufficient skills of the project team. In this case, a certain effort is required to compensate for such a situation, e.g., team training has to be conducted.

In CoBRA®, cost overhead is modeled by a so-called causal model. The causal model consists of factors affecting the costs of projects within a certain context. The causal model is obtained through expert knowledge acquisition (e.g., involving experienced project managers). An example is presented in Figure 2-1. The arrows indicate direct and indirect relationships. A sign (′+′ or ′–′) indicates the way a cost factor contributes to the overall project costs. The ′+′ and ′–′ represent a positive and negative relationship, respectively; that is, if the factor increases, the project costs will also increase (′+′) or decrease (′–′). For instance, if *Requirements volatility* increases, costs will also increase. One arrow pointing to another one indicates an interaction effect. For example, an interaction exists between *Disciplined requirements management* and *Requirements volatility*. In this case, increased disciplined requirements management compensates for the negative influence of volatile requirements on software costs.

The cost overhead portion resulting from indirect influences is represented by the second component of the sum shown in Equation 2.1.2. In general, CoBRA® allows for expressing indirect influences on multiple levels (e.g., influences on *Disciplined requirements management* and influences on influences thereon). However, in practice, it is not recommended for experts to rate all factors due to the increased complexity of the model and the resulting difficulties and efforts. Further details on computing the cost overhead can be found in [14].

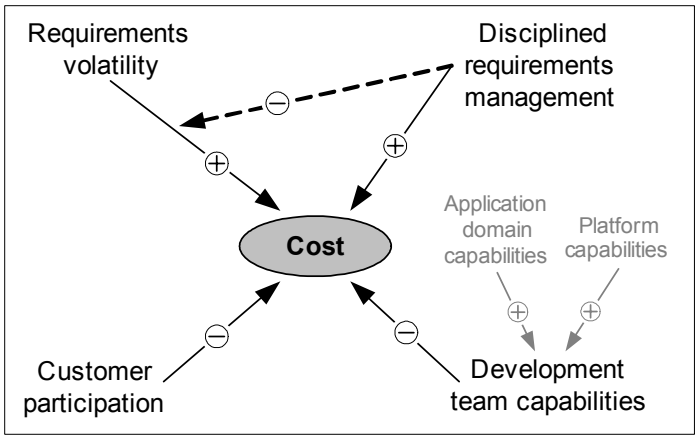

Figure 2-1: Example causal cost model

The influence on costs and between different factors is quantified for each factor using experts' evaluation. The influence is measured as a relative percentage increase of the costs above the nominal project. For each factor, experts are asked to give the increase of costs when the considered factor has the worst possible value (extreme case) and all other factors have their nominal values. In order to capture the uncertainty of evaluations, experts are asked to give three values: the maximal, minimal, and most likely cost overhead for each factor (triangular distribution).

The second component of CoBRA®, the nominal project costs, is based on data from past projects that are similar with respect to certain characteristics (e.g., development type, life cycle type) that are not part of the causal model. These characteristics define the context of the project. Past project data is used to determine the relationship between cost overhead and costs (see Equation 2.1.1). Since it is a simple bivariate dependency, it does not require much measurement data. In principle, merely project size and effort are required. The size measure should reflect the overall project volume, including all produced artifacts. Common examples include lines of code or Function Points [61]. Past project information on identified cost factors is usually elicited from experts.

Based on the quantified causal model, past project data, and current project characteristics, a cost overhead model (distribution) is generated using a simulation algorithm (e.g., Monte Carlo or Latin Hypercube). The probability distribution obtained could be used further to support various project management activities, such as cost estimation, evaluation of cost-related project risks, or benchmarking [14]. Figure 2-2 illustrates two usage scenarios using the cumulative cost distribution: calculating the project costs for a given probability level and computing the probability for exceeding given project costs.

Let us assume (scenario A) that the budget available for a project is 900 Units and that this project's costs are characterized by the distribution in Figure 2-2. There is roughly a 90% probability that the project will overrun this budget. If this probability represents an acceptable risk in a particular context, the project budget may not be approved. On the other hand, let us consider (scenario B) that a project manager wants to minimize the risks of overrunning the budget. In other words, the cost of a software project should be planned so that there is minimal risk of exceeding it. If a project manager sets the maximal tolerable risk of exceeding the budget to 30%, then the planned budget for the project should not be lower than 1170 Units.

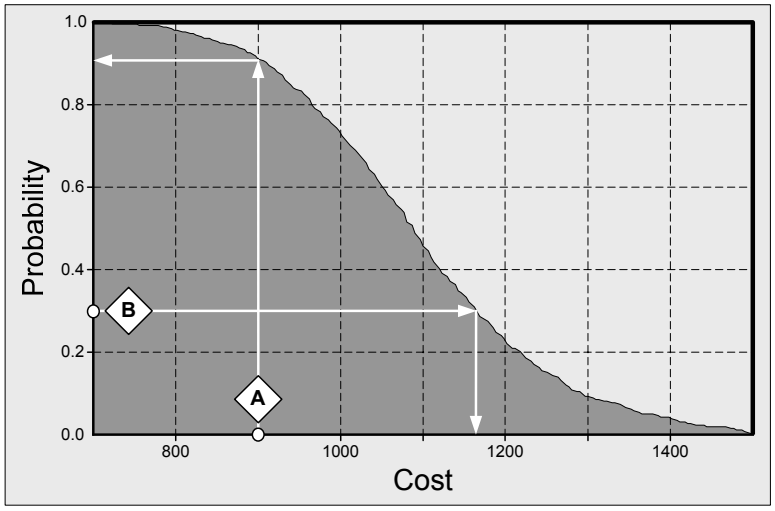

Figure 2-2: Example cumulative cost distribution

The advantage of CoBRA® over many other cost estimation methods is its low requirements with respect to measurement data. Moreover, it is not restricted to certain size and cost measures. The method provides means to develop an estimation model that is tailored to a certain organization's context, thus increasing model applicability and performance (estimation accuracy, consistency, etc.) A more detailed description of the CoBRA® method can be found in [14].

2.2 Simulation Techniques

Simulation is an approach to obtaining knowledge about the behavior or properties of a real system by creating and investigating a similar system or a mathematical model of the system.

Before computer simulations were possible, formally modeled systems had to be analytically solved to enable predicting the behavior of the system from a given number of parameters and initial conditions. Stochastic computer simulation methods like Monte Carlo sampling and Latin Hypercube sampling make it possible to handle more complex systems with larger numbers of input variables and more complex interactions between them.

2.2.1 Basic Principles of the Monte Carlo Method

The *Monte Carlo* method (MC) [91] is the most popular simulation approach, used in numerous science areas (see Section 3). One significant advantage of the MC method is the simplicity of its algorithmic structure. In principle, an MC algorithm consists of a process of producing random events (so-called *trials*). These events are used as input for the mathematical model and produce possible occurrences of the observed variable. This process is repeated n times (*iterations*) and the average over all occurrences is calculated as a result.

The mathematical foundation of the MC method is built upon the *strong law of large numbers* [60]:

Let η_i ($i \in N$) be a sequence of integrable, independent random variables with identical distributions and a finite expected value $\mu = E(\eta_i)$. It follows that for nearly any realization ω:

Equation 2.2.1
$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} \eta_i(\omega)}{n} = \mu$$

This means that we can transform η into the integral, stochastically, by taking repeated samples from η and averaging $\eta_i(\omega)$.

The *expected error* for n iterations can be estimated with [91]:

Equation 2.2.2 $$EAR = E(\left| \frac{1}{n} \sum_{i=1}^{z} \eta_i(\omega) - \mu \right|) \leq \frac{\sigma}{\sqrt{n}}$$

where σ is the standard derivation of the random variable η_i.

In this chapter, however, we also consider *sampling* as an attempt to approximate an unknown probability distribution with the help of randomly sampled values. In the context of MC, this means that we are not only interested in the expectation of the random variable (η), but also in an approximation of its distribution D_η, describable by its *probability density function* (pdf) *f(x)* [91].

The range of random samples obtained (result of independent realizations of η) build up an interval *[x,y]*. After breaking it into a number of equal-length intervals and counting the frequency of the samples that fall into each interval, we may construct a *histogram* that approximates the density distribution of the sampled random variable η (see Figure 2-3).

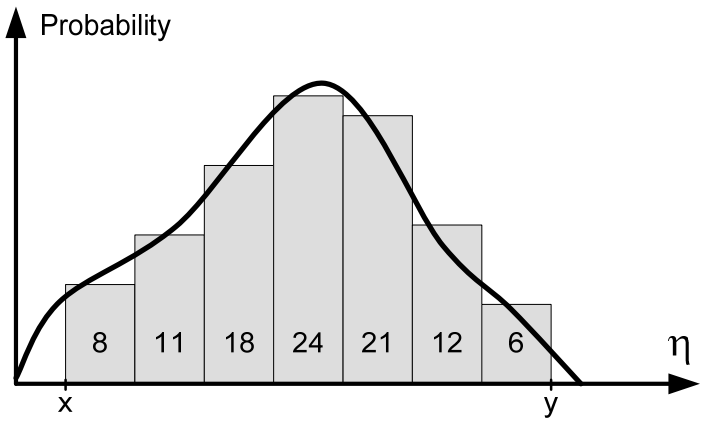

Figure 2-3: Example histogram of a probability variable

2.2.2 Stratification and Latin Hypercube Sampling

The expected error of the Monte Carlo method depends on the number of iterations n and the variance σ of the random variable η (see Equation 2.2.2). The error can thus be reduced either by increasing the number of trials and/or reducing the variance of the random variable. Numerous variance reduction techniques are discussed in the literature (see Section 3). One of them is stratification. In this section, we present the idea of stratification in general, as well as Latin Hypercube (LH) as a modification for multi-dimension problems that can be used without deeper knowledge of the random variable. We explain both: how LH can be used for mean computation and how it can be used for sampling.

When we use stratification for variance reduction, we break the domain of the input distribution into s independent domains (so-called *strata*). In a one-dimensional case, this is simply a partitioning of the range of the input distribution (Figure 2-4). If we use strata of equal size and consider the input variable ξ as being uniformly distributed across the half-open interval *[0,1)*, we can get random variables ξ_j for each j-th strata as:

Equation 2.2.3 $$\xi_j = \frac{j - \xi}{s}$$

So, we can simulate any random variable η with the density function *f(x)* if its inverse cumulative density $F^{-1}(x)$ is given:

Equation 2.2.4 $$\xi_S = \frac{1}{m \cdot s} \sum_{i,j=1}^{m,s} \eta_{i,j} = \frac{1}{n} \sum_{i,j=1}^{m,s} F^{-1}(\xi_j) \qquad \text{with } n = m \cdot s$$

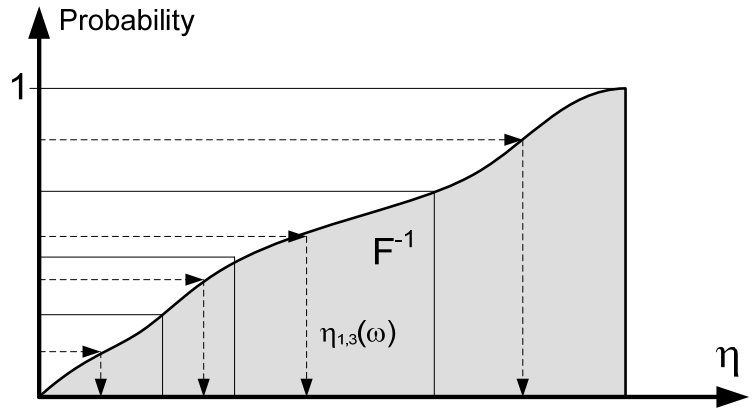

Figure 2-4: Example stratification for s and n = 4

Moreover, we obtain for m=1 a variance reduction of:

Equation 2.2.5 $$Var_{new} = Var_{old} - \sum_{j=1}^{n} \frac{(E - E_j)^2}{n^2}$$

where: E is the expectation of $F^{-1}(x)$ over [0,1] and

E_j is the expectation of $F^{-1}(x)$ over strata j.

In the multi-dimensional case, we have the problem that a straightforward generalization is not efficient [51]. The reason is that we must sample in too many cells (subintervals). If we want *s* strata in any dimension *d*, we need s^d cells to partition the input variable(s).

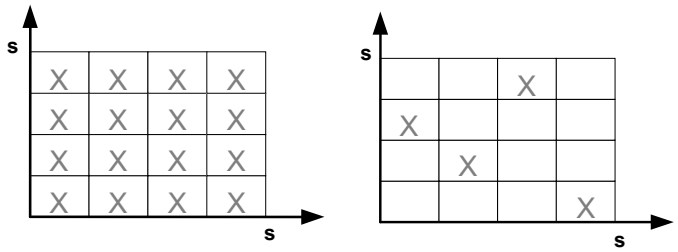

Figure 2-5: Two-dimensional straightforward application of stratification requires 16 cells to get 4 strata (left), the LH algorithm selects only 4 subintervals (right).

The *Latin Hypercube* technique (LH) [65] overcomes this problem by sampling each stratum only once. In order to obtain a perfect stratification within a single dimension, each randomly selected stratum is taken only once and independently of the strata within other dimensions. This can be guaranteed by using *d* independent permutations π_k of *{ 1,...,s }* with *k = 1,...,d*.

For $s = n$, we obtain the following estimator that converges against the expected value $E(\eta)$ (see also Equation 2.2.1):

Equation 2.2.6

$$\xi_{LH} = \frac{1}{n}\sum_{j=1}^{n}\eta_i = \frac{1}{n}\sum_{j=1}^{n}F^{-1}(\xi_{j,1},...,\xi_{j,d})$$

$$\text{with } \xi_{j,k} = \frac{\pi_k(j) - \xi}{n}$$

In order to sample a pdf using the LH approach, we can use the same procedure as for the MC approach (see Section 2.2.1). In fact, LH is an improvement (special case) of the MC approach. Therefore, the error can be estimated in the same way as for MC (see Section 2.2.1).

3. RELATED WORK

3.1 Software Effort Estimation Methods

3.1.1 Classification of Existing Effort Estimation Methods

Software researchers have made a number of attempts to systematize software effort estimation methods [104], [9], [8], [12].

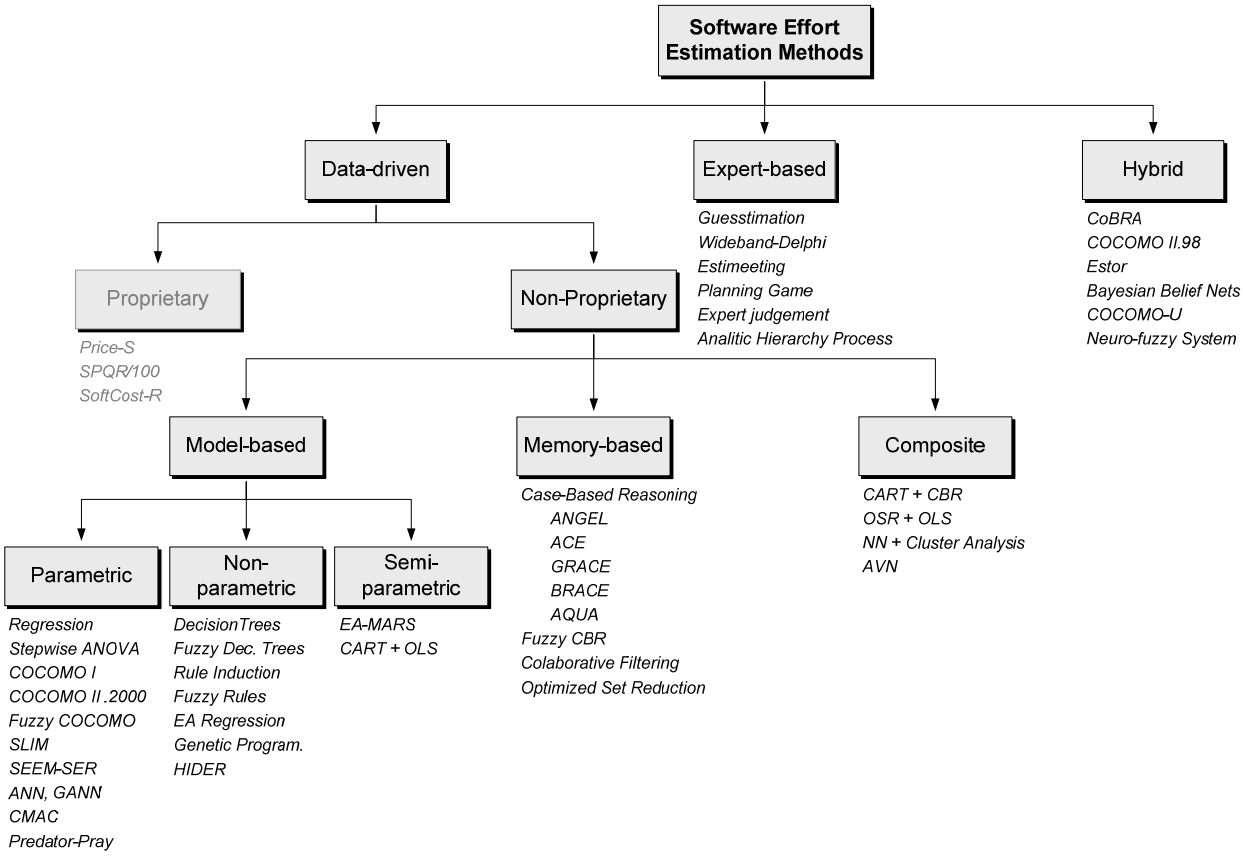

Figure 3-1: Classification of software effort estimation methods.

Proposed classification schemes are subjective and there is no agreement on the best one. Some of the classes, like *Price-to-Win*, cannot really be considered to be an estimation technique. Other classes are

not orthogonal, e.g., expert judgment can be used following a bottom-up estimation strategy. The systematization we propose in this chapter (Figure 3-1) is probably also not fully satisfactory, but is designed to overview the current status of effort estimation research and evaluate it against the most recent industrial requirements.

3.1.1.1 Data-driven Methods

Data-driven (data-intensive) methods refer to methods that provide effort estimates based on an analysis of measurement project data.

Proprietary vs. Non-proprietary

Proprietary effort estimation methods refer to methods that are not fully documented in the public domain. Examples of proprietary approaches are PRICE-S [19], SPQR/100 [45], and SoftCost-R [78]. Due to the lack of sufficient documentation, we exclude those methods from further consideration.

Model-based methods

Model-based methods provide a model that relates the dependent variable (effort) to one or more independent variables, typically a size measure and one or more effort factors. The model is built based on the historical project data and used for predictions; the data are generally not needed at the time of the prediction. An estimation model may be characterized by different parameters. They might be specified a priori before analyzing the project data (parametric methods) or determined completely from the data at the time of model development (non-parametric methods).

Parametric model-based methods specify the parameters of the estimation model a priori. ***Statistical regression*** methods, for instance, require a priori specification of the model's functional form and assume estimation errors to follow certain parametric distributions. Typical functional forms of the ***univariate regression*** include [55], [62], [83]: Linear, Quadratic, Cobb-Douglas, log-linear, and Translog. ***Multivariate regression***, on the other hand, constructs models that relate effort to many independent variables [67],[20]. Typical regression methods fit regression parameters to historical project data using ordinary least squares strategy. Yet, this technique is commonly criticized for its sensitivity to data outliers. As an alternative, robust regression has been proposed by several authors [68], [62]. Another statistical approach to handle unbalanced data is ***Stepwise Analysis of Variance*** (***Stepwise ANOVA***), which combines classical ANOVA with OLS regression [54]. In each step of an iterative procedure, ANOVA is applied to identify the most significant effort factor and remove its effect by computing the regression residuals; ANOVA is then applied again using the remaining variables on the residuals. Another critical point regarding classical regression is that it does not handle non-continuous (ordinal and categorical) variables. In order to solve that problem, generalized regression methods such as ***Categorical Regression*** [3] and ***Ordinal Regression*** [83] have been proposed.

Fixed-model estimation methods such as ***COCOMO*** [10], [8], ***SLIM*** [76], and ***SEER-SEM*** [41] also belong to the group of parametric methods. In fact, all three models have their roots in the early ***Jensen's regression models*** [42], [43]. All of them actually represent regression models that were once built on multiple proprietary project data and were intended to be applied "as is" in contexts that may significantly differ from the one they were built in. Yet, the software estimation community agrees that effort models work better when calibrated with local data [25], [67]. In order to improve the performance of fixed-model methods when applied within a specific context, various adjustment techniques have been proposed, such as periodical updates of fixed models to reflect current trends and changes in the software domain are (e.g., the family of COCOMO models [10], [8]). The frequency of official

model calibrations might, however, not be sufficient to keep them up to date [67]. Therefore, additional mechanisms to fit fixed models to local context have been suggested. Already, authors of fixed models are proposing certain *local calibration* techniques as part of a model's application procedures. Moreover, several adjustment approaches were proposed by independent researchers [90], [67]. The ***COCONUT*** approach, for instance, calibrates effort estimation models using an exhaustive search over the space of COCOMO I calibration parameters [67].

Recently, a parametric regression method that adapts ecological ***predator-prey models*** to represent dynamics of software testing and maintenance effort has been proposed [17]. The basic idea is that the high population of prey allows predators to survive and reproduce. In contrast, a limited population of prey reduces the chances of predators to survive and reproduce. A limited number of predators, in turn, creates favorable conditions for the prey population to increase. Authors adopt this phenomenon to software maintenance and testing. Software corrections represent predating software defects, and associated effort is fed by defects being discovered by the user. Perfective and adaptive maintenance are both fed by user needs, and the corresponding maintenance effort adapts itself to the number of change requests.

Parametric estimation also adapts selected machine learning approaches. Examples include ***Artificial Neural Networks*** (***ANN***). Network parameters such as architecture or input factors must be specified a priori; the learning algorithm then searches for values of parameters based on project data. The simplest ANN, so-called *single-layer perceptron*, consists of a single layer of output nodes. The inputs are fed directly to the outputs via a series of weights. In this way, it can be considered the simplest kind of feed-forward network. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above a certain threshold (typically 0), the neuron fires and takes the activated value (typically 1); otherwise, it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called McCulloch-Pitts neurons or threshold neurons. In the area of effort estimation, the *sigmoid* and *Gaussian* functions are usually used. Since single-unit perceptron are only capable of learning linearly separable patterns, multilayer perceptron are used to represent nonlinear functions. Examples of such ANN are the *Multilayer feed-forward back-propagation perceptron* (*MFBP*) [33] and the *Radial Basis Function Network* (*RBFN*) [33][88].

The ***Cerebellar Model Arithmetic Computer*** (***CMAC***), also known as *Albus perceptron*, is a special type of a neural network, which represents a multidimensional function approximator [82]. It discretizes values of continuous input to select so-called training points. CMAC first learns the function at training points and then interpolates to intermediary points at which it has not been trained. The CMAC method operates in a fashion similar to a lookup table, using a generalization mechanism so that a solution learned at one training point in the input space will influence solutions at neighboring points.

Finally, ***Evolutionary Algorithms*** (***EA***) represent an approach that can be used to provide various types of traditionally data-based estimation models. EA simulates the natural behavior of a population of individuals (chromosomes) by following an iterative procedure based on selection and recombination operations to generate new individuals (next generations). An individual (chromosome) is usually represented by a finite string of symbols called chromosome. Each member of a population encodes a possible solution in a given problem search space, which is comprised of all possible solutions to the problem. The length of the string is constant and completely dependent of the problem. The finite string of symbol alphabet can represent real-valued encodings [2],[80], tree representation [21], or software code [84]. In each simulation iteration (generation), relatively good solutions produce offspring that replace relatively worse ones retaining many features of their parents. The relative quality of a solution is based on the

fitness function, which determines how good an individual is within the population in each generation, and what its chance of reproducing is, while others (worse) are likely to disappear. New individuals (offspring) for the next generation are typically created by using two basic operations, crossover and mutation.

Shukla, for instance, used evolutionary algorithms to generate an optimal artificial neural network for a specific problem [89]. The method was called a ***Neuro-genetic Effort Estimation Method*** (***GANN***). The author encoded the neural network using so-called strong representation and learned it using genetic algorithm. The ANN generated in each cycle was evaluated against a fitness function, defined as the inverse of the network's prediction error when applied on the historical project data.

Non-parametric model-based methods differ from parametric methods in that the model structure is not specified a priori but instead is determined from quantitative (project) data. In other words, non-parametric estimators produce their inferences free from any particular underlying functional form. The term nonparametric is not meant to imply that such methods completely lack parameters, but rather that the number and nature of the parameters are flexible and not fixed in advance.

Typical non-parametric methods originate from the machine learning domain. The most prominent examples are ***decision tree*** methods. The most popular one in the artificial intelligence domain, the C4.5 algorithm has not been exploited much in the software effort estimation domain due to its inability to handle numerical input data [2]. Proposed in [11], the classification and regression trees (CART) method overcame this limitation and is applicable for both categorical and numerical data. The CART algorithm was re-implemented in numerous software tools such as CART [11], CARTX [94], and GC&RT [97]. Decision trees group instances (software projects) with respect to a dependent variable. A decision tree represents a collection of rules of the form: *if (condition 1 and ...and condition N) then Z* and basically forms a stepwise partition of the data set being used. In that sense, decision trees are equivalent to decision rules, i.e., a decision tree may be represented as a set of decision rules. Successful applications in a number of different domains have motivated software researchers to apply dedicated ***rule induction*** methods (RI) that generate effort estimation decision rules directly from statistical data [64].

Finally, some applications of the evolutionary algorithms can be classified as non-parametric, for instance, EA applied to generate regression equations (***EA Regression***) [21], [16]. In that case, the elements to be evolved are trees representing equations. That is, the population P of the algorithm is a set of trees to which the crossover and mutation operators are applied. A *crossover* exchanges parts of two equations, preserving the syntax of the mathematical expression, whereas a *mutation* randomly changes a term of the equation (function, variable or constant). The terminal nodes are constants or variables, and not-terminals are basic functions that are available for system definition. Each member of the population (regression equation) is evaluated with respect to the fitness function defined as the average estimation error on the historical project data obtained when using the member.

Shan et al. applied so-called ***Grammar Guided Genetic Programming*** (***GGGP***) to generate computer programs that estimate software development effort [84]. Generating computer programs using EA creates a separate area of machine learning called ***Genetic Programming*** (***GP***). The objective of GP is to optimize a population of computer programs according to a fitness function determined by a program's ability to perform a given computational task (in this case accurately estimate software effort). In GGGP program generation process is guided by a specific language grammar that (1) imposes certain syntactical constraints, and (2) incorporates background knowledge into the generation process. The grammar

used resulted in models that are very similar to regression. The means square estimation error on the historical project data was applied as the fitness function to evaluate the model-generated GGGP process.

Finally, Aguilar-Ruiz et al. applied evolutionary algorithms to generate a set of ***Hierarchical Decision Rules*** (***HIDER***) for the effort classification problem [2]. Members of the population are coded using a vector of real values. The vector consists of pair values representing an interval (lower and upper bound) for each effort factor. The last position in the vector represents a discrete value of effort (effort class). The EA algorithm extracts decision rules from this representation. Each member of the population (decision rule) is evaluated with respect to the fitness function that discriminates between correct and incorrect effort predictions (classifications) of historical projects using the member.

Semi-parametric model-based methods represent methods that contain both parametric and non-parametric components. In a statistical sense, a semi-parametric method produces its inferences free from a particular functional form but within a particular class of functional forms, e.g., it might handle any functional form within the class of additive models.

A typical example of the semi-parametric method where parametric and non-parametric elements were merged is the *integration of decision trees and ordinary least squares regression* (***CART+OLS***) proposed by Briand and colleagues [13], [12]. The method generates a CART tree and applies regression analysis to interpret projects at each terminal node of the tree.

The application of evolutionary algorithms to generate ***Multiple Regression Splines*** (***MARS***) is another example of the semi-parametric method [80]. Instead of building a single parametric model, ***EA-MARS*** generates multiple parametric models, using linear regression. In this case, the search space comprises a set of cut-points (CP) in the independent variable (e.g., software size measured in function points FP), so a different parametric estimation model can be used for the intervals that comprise such cut-points.

Memory-based Methods

Model-based methods, such as neural networks or statistical regression, use data to build a parameterized model (where parameters must be specified a priori or might be learned from data). After training, the model is used for predictions and the data are generally not needed at the time of prediction. In contrast, *memory-based* methods (or *analogy-based methods*) do not create a model but explicitly retain the available project data, and use them each time a prediction of a new project needs to be made. Each time an estimate is to be provided for a new project (target), the project data (case base) is searched for the projects (analogues) that are most similar to the target. Once the most similar projects are found, their actual efforts are used as a basis for estimation.

The ***Case-based Reasoning*** (***CBR***) method represents an exact implementation of the memory-based estimation paradigm. It estimates a new project (target) by adapting the effort of the most similar historical projects (analogues) [50].

In order to find project analogies, the CBR method first selects the most relevant characteristics of a software project and defines a project similarity measure upon it. The early methods, such as the ***Analogy Software Tool*** (***ANGEL***), required a priori selection of the most significant characteristics or simply used all characteristics available in the data repository [87]. Later, automatic factor selection [50] and weighting [58] techniques were applied that optimize the performance of the CBR estimator. The most commonly used similarity measure is based on the *Euclidian distance* [87]. The ***Gray Relational Analysis Based Software Project Effort*** (***GRACE***) method computes the *gray relational grade* [92], and the ***Collaborative Filtering*** (***CF***) method uses a similarity measure proposed in the field of information

retrieval to evaluate the similarity between two documents [72]. The most recent ***AQUA*** method [58] uses a locally weighted similarity measure, where local weights are computed for different project attributes dependent on their type (nominal, ordinal, continuous, set, fuzzy, etc.) The ***Analogical and Algorithmic Cost Estimator*** (***ACE***) and ***Bootstrap Based Analogy Cost Estimation*** (***BRACE***) consider several alternative similarity measures. ACE uses average similarity computed over several distance measures [104], [103], whereas BRACE analyzes each alternative similarity measure to select the one that entails optimal estimation performance [95]. In addition, scaling techniques are used to transform values of project characteristics such that all have the same degree of influence, independently of the choice of units [87], [72], [92]. Based on the results of the similarity analysis, analogues are selected to base target estimates on. One group of methods proposes a small constant number (1 to 3) of nearest neighbors [87], [103], [13], [66]. Other methods, such as BRACE [95] and AQUA [58], determine the optimal number of analogies in a cross-validation analysis. The effort of selected analogues is then adapted to predict the target project. In case of a single analogue, its effort may be adopted [13] or additionally adjusted using the size of the analogue and target projects [103], [95]. Adapting several analogues includes median, mean, distance-weighted mean [87], [66], [72], and inverse rank weighted mean [50], [66]. Again, BRACE analyzes several alternative adaptation approaches to determine the optimal one [95].

Another way of implementing memory-based estimation is represented by the ***Optimized Set Reduction*** (***OSR***) method [13] [105]. Based on the characteristics of the target project, OSR iteratively partition the set of project data into subsets of similar projects that assure increasing information gain. Similarly to decision trees, the final effort estimate is based on projects in the terminal subset.

Composite methods integrate elements of the model- and memory-based methods. Typical examples are applications of CBR or OLS regression to adapt projects in the CART terminal node [13], [12] or OSR terminal subset [48]. Another example is the usage of *cluster analysis* to group similar projects in order to facilitate the training of the neural network [57]. The ***analogy with virtual neighbor method*** (***AVN***) represents a more sophisticated combination of model- and memory based methods. It enhances the classical CBR approach in that the two nearest neighbors of a target project are identified based on the normalized effort, which is computed from the multiple-regression equation. Moreover, AVN adjusts the effort estimation by compensating for the location of the target project relative to the two analogues.

3.1.1.2 Expert-based Methods

Expert-based estimation methods are based on the judgment of one or more human experts. The simplest instance of the unstructured expert-based estimation is the so-called ***Guesstimation*** approach, where a single expert provides final estimates. An expert could just provide a guess ("*rule-of-thumb*" method) or give estimates based on more ***structured reasoning***, for example, breaking the project down into tasks, estimating them separately and summing those predictions into a total estimate (bottom-up approach). An example formal way of structuring expert-based effort estimation is represented by the ***Analytic Hierarchy Process*** (***AHP***) method, which systematically extracts a subjective expert's predictions by means of pair-wise comparison [85].

Due to many possible causes of bias in individual experts (optimist, pessimist, desire to win, desire to please, political), it is preferable to obtain estimates from more than one expert [69]. A group of experts may, for instance, provide their individual estimates, which are then aggregated to a final estimation, e.g., by use of statistical mean or median. This is quick, but subject to adverse bias by individual extreme estimates. Alternative *group consensus approaches* try to hold group meetings in order to obtain

expert agreement with regard to a single estimate. Examples of such methods are ***Wide-band Delphi*** [10], [69], ***Estimeeting*** [99], and, recently defined in the context of agile software development, ***Planning Game*** [7]. A more formal approach to integrating uncertain estimates of multiple human experts is ***Stochastic Budgeting Simulation*** (***SBS***) [22]. SBS employs random sampling to combine effort of individual effort items (work products or development activities) and project risks specified by experts in terms of triangular or Erlang distribution.

3.1.1.3 Hybrid Methods

Hybrid methods combine data- and expert-based methods. In practice, hybrid methods are perceived as the answer to the more and more common observation that human experts, when supported by low-cost analytical techniques, might be the most accurate estimation method [44].

The ***ESTOR*** method, for instance, [70] provides the initial effort of a target project based on the CBR estimation and then adjusts it by applying a set of expert-based rules, which account for the remaining differences between the analog and the target project. The ***Cost Estimation, Benchmarking and Risk Analysis*** (***CoBRA***) method applies expert-based effort causal modeling to explain the variance on the development production rate measured as effort divided by size [14], [100].

Recently, the software research community has given much attention to *Bayes Theorem* and ***Bayesian Belief Networks*** (***BBN***). Chulani, for instance, employed Bayes theorem to combine a priori information judged by experts with data-driven information and to calibrate one of the first versions of the COCOMO II model, known as ***COCOMO II.98*** [8]. Several researchers have adapted ***BBN*** to build causal effort models and combine a priori expert judgments with a posteriori quantitative project data [24], [75]. Recently, BBN was used to construct the probabilistic effort model called ***COCOMO-U***, which extends COCOMO II [8] to handle uncertainty [108].

3.1.2 Handling Uncertainty

Uncertainty is inherent to software effort estimation [53][47]. Yet, software managers usually do not understand how to properly handle the uncertainty and risks inherent in estimates to improve current project budgeting and planning processes.

Kitchenham et al. conclude that estimation, as an assessment of a future condition, has inherent probabilistic uncertainty, and formulate four major sources of estimation uncertainty: measurement error, model error, assumption error, and scope error [53]. We claim that limited cognition of effort dependencies is another major source of estimation uncertainty. We distinguish two major sources of effort estimation uncertainty: probabilistic and possibilistic. ***Probabilistic uncertainty*** reflects the random character of the underlying phenomena, i.e., the variable character of estimation problem parameters. Uncertainty is considered here in terms of probability, i.e., by the random (unpredictable, non-deterministic) character of future software project conditions, in particular, factors influencing software development effort. ***Possibilistic uncertainty*** (*epistemological uncertainty*) reflects the subjectivity of the view on modeled phenomena due to its limited cognition (knowledge and/or experience). This may, for instance, include limited granularity of the description of the modeled phenomena, i.e., a finite number of estimation parameters (e.g., effort factors and the ways they influence effort). The lack of knowledge may, for instance, stem from a partial lack of data, either because this data is impossible to collect or too expensive to collect, or because the measurement devices have limited precision. Uncertainty is considered here in terms of possibilities.

Handling Probabilistic Uncertainty

The most common approach, which actually explicitly refers to probabilistic uncertainty, is based on the analysis of the probability distribution of given input or output variables. Representing effort as a distribution of values contributes, for instance, to better understanding of estimation results [53]. Motivated by this property, a number of estimation methods that operate on probability distributions have been proposed. Recent interest in *Bayesian Belief Networks* [24], [75], [108] is one example of this trend. Another one is generally acknowledged to be the application of *random sampling* over the multiple estimation inputs represented by probability distribution [35], [14], [22], [76], [95], [100]. The resulting effort distribution may then be interpreted using various analysis techniques. Confidence and prediction intervals are commonly used statistical means to reflect predictive uncertainty. Human estimators are traditionally asked to provide predicted min-max effort intervals based on given confidence levels, e.g., "almost sure" or "90 percent confident". This approach may, however, lead to overoptimistic views regarding the level of estimation uncertainty [46]. Jørgensen proposes an alternative, so-called *pX-effort approach*, where instead of giving a number, the human estimator should give ranges and views based on probabilistic distributions [46]. The pX-view means there is an X percent probability of not exceeding the estimate.

Handling Possibilistic Uncertainty

In practice, handling possibilistic uncertainty involves mainly the application of the ***fuzzy set theory*** [109]. The approaches proposed in the effort estimation domain can be principally classified into two categories: (1) fuzzifying existing effort estimation methods and (2) building specific rule-based fuzzy logic models.

The first approach simply adapts existing software effort estimation methods to handle the uncertainty of their inputs and outputs using fuzzy sets. A typical estimation process consists of three steps: (1) *fuzzification* of inputs, (2) imprecise reasoning using fuzzy rules, and (3) *de-fuzzification* of outputs. Inputs and outputs can be either linguistic or numeric. Fuzzification involves finding the membership of an input variable with a linguistic term. The membership function can, in principle, be either defined by human experts or analytically extracted from data, whereas fuzzy rules are usually defined by experts. Finally, de-fuzzification provides a crisp number from the output fuzzy set. One example implementation of such an idea is a series of fuzzy-COCOMO models [71], [39], [59], [1]. Other adaptations of traditional estimation methods include fuzzyfying similarity measures within CBR methods [38], [58] or information gain measure in decision trees [36].

Rule-based fuzzy logic models directly provide a set of fuzzy rules that can then be used to infer predictions based on the fuzzy inputs [63]. Fuzzy rules may be based on human judgment [62] or learned from empirical data [107]. ***Neuron-fuzzy systems*** (***NFS***) represent a hybrid approach to learn fuzzy rules. The equivalence between neural networks (ANN) and fuzzy logos systems makes it possible to create initial fuzzy rules based on expert judgment, translate them into equivalent ANN, and learn weights for the rules from quantitative data [27]. ***Neuro-Fuzzy COCOMO*** employs NFS to individually model each linguistic input of the COCOMO model [37].

Another approach to handle possibilistic uncertainty is based on the ***rough sets theory*** [74]. A rough set is a formal approximation of a crisp set (i.e., conventional set) in terms of a pair of sets, which give the lower and the upper approximation of the original set. The lower and upper approximation sets themselves are crisp sets in the standard version of the rough sets theory. The rough sets theory was recently applied within the AQUA+ method to cover the uncertain impact of effort drivers on effort [58].

3.1.3 Evaluation and Comparison of Existing Effort Estimation Methods

Literature on software effort estimation methods is rich with studies that evaluate and compare several estimation methods [12], [47], [69], [49]. However, they do not provide a clear answer to the question "Which method is the best one"?

The first impression one may get when looking over hundreds of empirical studies published so far, is that the only reasonable criterion to evaluate an estimation method is the accuracy of the estimates it derives. The second impression is that this criterion is probably not very helpful in selecting the best estimation method because instead of at least converging results, the reader often has to face inconsistent and sometimes even contradicting outcomes of empirical investigations [47]. The source of this apparent inconsistency is a number of factors that influence the performance of estimation methods, but which are usually not explicitly considered in the published results. Examples are: inconsistent empirical data (source, quality, preprocessing steps applied, etc.), inconsistent configuration of apparently the same method, or inconsistent design of the empirical study (e.g., evaluation strategy and measures).

Despite a plethora of published comparative studies, software practitioners have actually still hardly any basis to decide which method they should select for their specific needs and capabilities. In this chapter we propose an evaluation framework derived directly from industrial objectives and requirements with respect to effort estimation methods[3]. In addition, the definitions of individual criteria are based on related works [10], [16], [12], [1], [9], [52].

We propose to evaluate individual criteria using the 4-grade Likert scale [93]. For that purpose, we define each criterion in the form of a sentence that describes its required value from the perspective of industrial objectives and capabilities. We grouped the evaluation criteria in the two groups: criteria related to the estimation method (C01 to C10) and criteria related to the estimation outputs (C11 to C13).

> ***C01.*** *Expert Involvement*: The method does not require extensive expert's involvement, i.e., it requires a minimal amount of experts, limited involvement (effort) per expert, minimal expertise.
>
> ***C02.*** *Required data*: The method does not require many measurement data of a specific type (i.e., measurement scale) and distribution (e.g., normal).
>
> ***C03.*** *Robustness*: The method is robust to low-quality inputs, i.e., incomplete (e.g., missing data), inconsistent (e.g., data outliers), redundant, and collinear data.
>
> ***C04.*** *Flexibility*: The method is free from a specific estimation model and provides context-specific outputs.
>
> ***C05***. *Complexity*: The method has limited complexity, i.e., it does not employ many techniques, its underlying theory is easy to understand, and it does not require specifying many sophisticated parameters.
>
> ***C06.*** *Support level*: There is comprehensive support provided along with the method, i.e., complete and understandable documentation as well as a useful software tool.
>
> ***C07.*** *Handling uncertainty*: The method supports handling the uncertainty of the estimation (i.e., inputs and outputs).

[3] A series of industrial surveys were performed as part of the research presented in this section. The presentation of the detailed results is, however, beyond the scope of this chapter and will be published in a separate article.

C08. *Comprehensiveness*: The method can be applied to estimate different kinds of project activities (e.g., management, engineering) on various levels of granularity (e.g., project, phase, task).

C09. *Availability*: The method can be applied during all stages (phases) of the software development lifecycle.

C10. *Empirical evidence*: There is comprehensive empirical evidence supporting theoretical and practical validity of the method.

C11. *Informative power*: The method provides complete and understandable information that supports the achievement of numerous estimation objectives (e.g., effective effort management). In particular, it provides context-specific information regarding relevant effort factors, their interactions, and their impact on effort.

C12. *Reliability*: The method provides the output that reflects the true situation in a given context. In particular, it provides accurate, precise and repeatable estimation outputs.

C13. *Portability*: The method provides estimation outputs that are either applicable in other contexts without any modification or are easily adaptable to other contexts.

We evaluate existing estimation methods on each criterion by rating our agreement regarding the extent to which each method fulfills a given requirement. We use the following symbols: (++) strongly agree, (+) agree, (-) disagree, (--) strongly disagree. Presented in the Table 3-1 evaluation is based on individual author's experiences and critique presented in related literature. It includes both subjective and objective results.

An evaluation of existing estimation methods against industry-oriented criteria shows that none of them fully responds to industrial needs. One of the few methods that support most of the estimation objectives is the CoBRA method. Yet, it has still several significant weaknesses that may prevent its wide industrial acceptance. First is the substantial involvement of human experts. The CoBRA causal effort model (the so-called cost overhead model) is currently built exclusively by experts, which significantly increases estimation costs and reduces the reliability of outputs. The second problem is the unclear impact of the selected simulation technique on the reliability and computational cost of estimation outputs. In this chapter, we investigate the second problem.

		Estim. Method	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13
Data-driven	***Model-based***	Uni-regression	++	-	--	-	++	++	-	++	++	++	--	+	-
		Multi-Regression	++	--	--	+	+	++	-	++	++	++	-	+	-
		CATREG	++	--	--	+	+	+	-	++	++	-	-	+	-
		Ordinal Regression	++	--	--	+	+	+	+	++	++	-	-	+	-
		S-ANOVA	++	+	+	+	+	+	-	++	++	-	-	+	-
		COCOMO I	++	-	-	--	+	++	-	-	-	++	-	-	+
		COCOMO II	++	-	-	--	+	++	-	-	+	++	-	-	+
		SLIM	++	-	-	--	-	++	-	-	+	-	-	+	+
		SEER-SEM	++	-	-	--	-	+	-	-	+	--	-	+	+
		ANN	++	-	+	+	--	++	--	++	++	+	-	+	-
		Decision Trees	++	+	+	++	-	+	-	++	++	+	++	+	+
		Rule Induction	++	+	+	++	-	+	--	++	++	-	+	+	+
		CMAC	++	-	+	+	--	-	--	++	++	-	-	-	-
		EG/GP	++	++	+	++	-	+	--	++	++	+	+	-	+
		CART+OLS	++	-	+	++	-	+	-	++	++	-	++	+	+
	Memory-based	OSR	++	+	++	++	--	+	-	++	++	+	++	-	+
		ANGEL	++	+	+	++	+	++	--	++	++	++	-	+	-
		GRACE	++	+	+	++	+	+	--	++	++	+	-	+	-
		BRACE	++	+	+	++	+	+	+	++	++	+	-	-	-
		AQUA	++	+	++	++	-	+	--	++	++	+	-	++	-
		CF	++	+	+	++	+	+	--	++	++	-	-	+	-
	Composite	CART+CBR	++	+	+	++	+	+	-	++	++	-	++	-	+
		OSR+OLS	++	+	+	++	--	+	-	++	++	-	++	-	+
		ANN+Clustering	++	-	+	+	--	+	--	++	++	-	-	+	-
		AVN	++	+	++	-	+	+	-	++	++	-	-	++	-
		ACE	++	+	++	++	+	+	--	++	++	-	-	+	-
Expert-based		Guesstimation	-	++	+	++	++	--	-	+	-	+	-	--	-
		Wideband Delphi	--	++	++	++	++	++	-	+	+	-	-	-	-
		Estimeeting	--	++	++	++	++	-	-	+	+	-	-	-	-
		AHP	-	++	++	++	+	+	-	+	+	-	-	-	+
		SBS	--	++	++	++	+	-	+	+	+	-	-	+	-
		Planning Game	--	++	++	++	++	++	+	+	+	+	-	-	+
Hybrid		COCOMO II.98	-	-	+	--	+	+	+	--	+	-	-	+	+
		CoBRA	-	+	++	++	+	++	++	++	++	+	++	++	+
		BBN	-	+	+	++	+	++	++	++	++	-	++	-	+
		ESTOR	+	-	-	++	+	-	-	++	++	-	-	-	-
		NFS	++	-	+	+	--	++	+	++	++	+	-	-	+

Table 3-1 Evaluation and comparison of existing software effort estimation methods

3.2 Overview of Random Sampling Techniques

In Section 2.2, we provided the necessary theoretical foundations of simulation techniques such as Latin Hypercube (LH) and Monte Carlo (MC), including literature references. Therefore, we reduce the scope of our review in this section to literature regarding comparative studies of sampling techniques (concerning their theoretical and practical efficiency/usability issues) that can be applied in the context of cost estimation with CoBRA®. We consider only methods that can be used for high problem dimensions (number of input distributions). We do not consider approaches for "very high-dimensional" problems like Latin Supercube Sampling [73], since they are of limited practical applicability in the context of software cost estimation.

Saliby describes and compares in [81] a sampling method called Descriptive Sampling (DS) against Latin Hypercube (LH) and shows that DS is an improvement over LH. He proves that DS represents the upper limit of maximal improvement over standard MC that could be achieved with LH. Yet, he also showed that the improvement of DS over LH sampling decreases with an increasing number of iterations. The experimental data provided in the chapter shows that the improvement at even relatively small numbers of iterations, such as 1,000 (in our study we used between 10,000 and 640,000 iterations) is not significant anymore (<0.4%). Moreover, five years later, Saliby [81] compared the performance of six Monte Carlo sampling methods: Standard Monte Carlo, Descriptive Sampling, Latin Hypercube, and Quasi-Monte Carlo using three different numeric sequences (Halton, Sobol, Faure). Their convergence rates and precision were compared with the standard MC approach in two finance applications: a risk analysis and a correlated stock portfolio evaluation. In this study, the best aggregate performance index was obtained by LH sampling.

Additionally, there exist a lot of variance reduction techniques (VRT) that promise to improve the standard Monte Carlo method. A collection of them is presented in [56]. In contrast to MC, DS and LH, (1) these techniques require knowledge of the function to be estimated, (2) they can be used only under special circumstances, and/or (3) they require an appropriate adoption of the algorithm dependent on the function to be estimated.

Although for our context (the context of software project cost simulation), no results could be found in the literature, LH is being used successfully in other domains, e.g., in the automotive industry, for robustness evaluations [110] and in the probability analysis of long-span bridges [31]. In [79], it is mentioned that standard MC often requires a larger number of samples (iterations) to achieve the same level of accuracy as an LH approach. These and other articles show us the practical applicability of LH in various domains. Therefore, we see LH sampling as a promising approach to improving the standard MC method. Nevertheless, its concrete performance improvement depends on the application area with the associated simulation equations and input distributions.

Data about LH on a more theoretical level is provided by [34]. Here, standard MC and LH sampling are compared in an experiment where two simple synthetic functions, one monotonic and one non-monotonic, are sampled with both approaches. For both functions and for 25, 50 and 100 iterations, authors showed that the samples produced by the LH approach were more stable. Additionally, Owen showed in 1997 [73] that an LH sampling with n iterations never has worse accuracy than a standard MC sampling with $n-1$ iterations.

Avramidis et al. [1] investigate various techniques to improve estimation accuracy when estimating quantiles with MC approaches. The results of this work were of special interest for us because our aim of sampling a probability function has more in common with quantile estimation (see DoP in Section

7.1.1) than with the estimation of its mean (which is the common case). Avramidis provides a theoretical proof that for the class of probability functions with a continuous and nonzero density function, the LH approach delivers at least equally accurate (or more accurate) results than the standard MC method. In a second work [5], Avramidis also provides experimental evidence that supports this conclusion in the context of the upper extreme quantiles estimation of the network completion time in a stochastic activity network.

4. PROBLEM STATEMENT

The problem described in the introduction is to evaluate whether more accurate and efficient techniques than standard Monte Carlo (MC) exist that support software cost estimation and cost risk management. CoBRA® was selected as a representative method for this evaluation. Further, the extent of accuracy improvement above standard MC should be determined.

More precisely, we want to determine and compare the accuracy and efficiency of methods that transform the input of CoBRA® project cost simulation (i.e., the expert estimations) into the data required for the different kinds of predictions provided by CoBRA®.

In the following, fundamental research scenarios are described. Afterwards, research questions are stated based on the problem statement.

The following three scenarios cover all principle kinds of application of simulation data in the CoBRA® method. Even through they are derived from the typical application of the CoBRA® method, we see no reason why they should not be general scenarios when applying any cost estimation or cost risk analyzing methods:

AS1: The question to be answered in this scenario is: How much will the project cost? The answer should be the point estimation of the project cost, a single value that represents the average project costs. Spoken in terms of probability theory, we look for the expected value of the project costs. This value is also needed to validate the accuracy of a built CoBRA® model with the help of statistics.

AS2: The question to be answered in this scenario is: How high is the probability that the project costs stay below a given budget of X? It is possible to answer this question for any X if a probability distribution of the project costs is available. In this case, we can simply calculate P(Cost<X).

AS3: The question to be answered in this scenario is: Which is a realistic budget for a given project that is not exceeded with a probability of Y? Considering the previous scenario, this is finally the inverse application of the project cost probability distribution; here, we are interested in X with P(Cost<X)=Y, where Y is given.

When we compare the scenarios, we see that AS2 and AS3 make use of a variable to allow customization. Therefore, they require the knowledge of the entire cost probability distribution, whereas AS1 requires only a single characterizing value of the distribution, the expectation value. As mentioned above, the study focuses on the transformation of the simulation input (i.e., consolidated expert estimations for the different influencing factors) into the data required in the different application scenarios. This is the transformation that was done till now by standard Monte Carlo sampling. The *input* is a matrix $D^{(m,n)}$ of estimates of m different experts for n different cost drivers (see Figure 4-1). The estimates themselves are triangular distributions that represent the expert opinion as well as the uncertainty in their answers [102]. The output for AS1 should be the (approximated) expected value; the *output* for AS2 and AS3 should be the (approximated) probability distribution for further calculations based on variable X or Y (see Figure 4-2, for example).

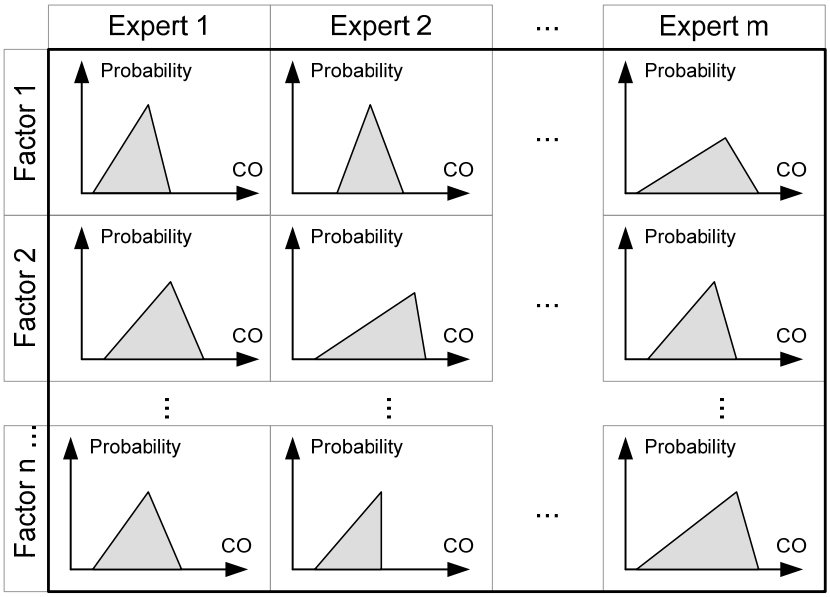

Figure 4-1: Example of an expert estimation matrix $D^{(m,n)}$

Based on this more detailed description of the problem and the application scenarios that should be considered, we derived the following research questions:

RQ1: Do feasible (in terms of calculation effort) analytical solutions exist to calculate data required in AS1, AS2, and AS3?

RQ2: If data cannot be calculated analytically, do stochastic solutions exist that calculate the data used in AS1, AS2, and AS3 more accurately and more efficiently than the standard Monte Carlo simulation approach?

RQ3: How high is the average accuracy and efficiency gained by choosing a certain simulation approach (e.g., Latin Hypercube) when compared with standard Monte Carlo sampling?

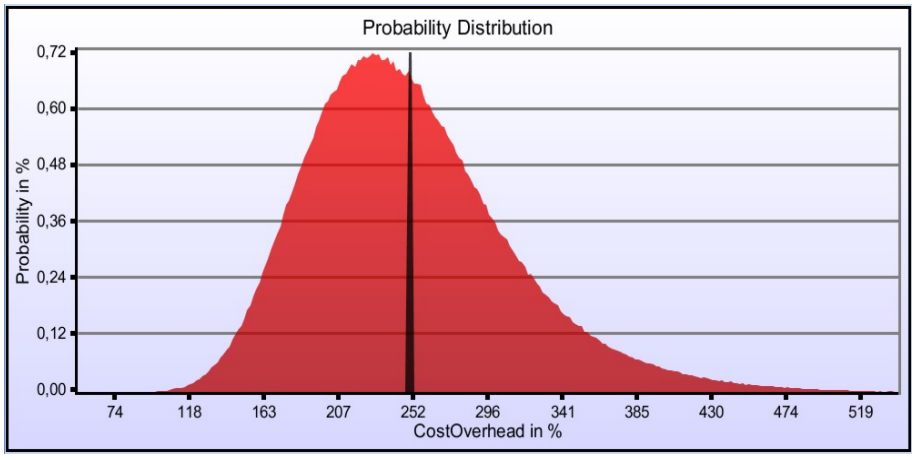

Figure 4-2: Example cost overhead distribution.

In the next section, we answer RQ1 and explain why we need stochastic approaches for AS2 and AS3. Then, Section 6 presents the adoption of stochastic approaches to the context of CoBRA® and explains the need to conduct an experiment to answer RQ2 and RQ3. The planning of the experiment, its execution, and its results are described in Section 7. Finally, Section 8 summarizes the answers to all research questions.

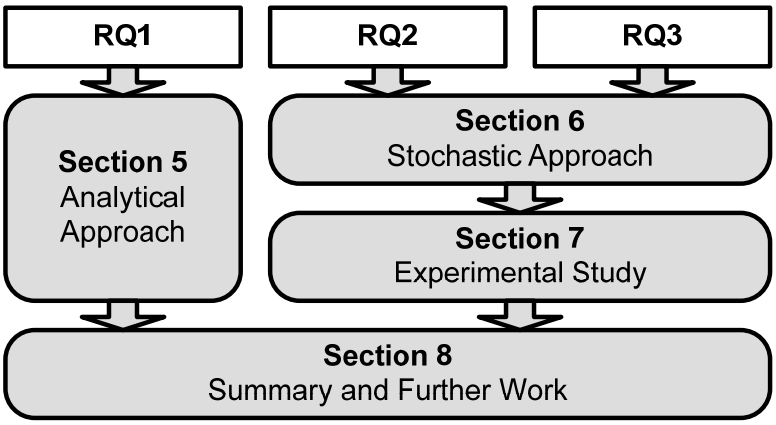

Figure 4-3: Structure of chapter.

5. ANALYTICAL APPROACHES

In the following, we show that the data that have to be provided in application scenario AS1 (i.e., expected project costs) can be calculated analytically. However, we also show that the analytical derivation of the cost probability distribution necessary in AS2 and AS3 is not feasible with respect to the calculation effort.

5.1 Point Estimation

To calculate the expected value analytically, in a first step, we have to derive a mathematical representation of the cost overhead distribution D described by the $D^{(m,n)}$ matrix. Having a distribution $D_{e,v}$ that represents a distribution provided by expert e for the cost factor v, we obtain the following equation for distribution D:

Equation 5.1.1

$$D = \sum_{v=1}^{n} D_v \text{ with } D_v = \begin{cases} D_{1,v} \text{ with probability } \frac{1}{m} \\ D_{2,v} \text{ with probability } \frac{1}{m} \\ \dots \\ D_{m,v} \text{ with probability } \frac{1}{m} \end{cases}$$

However, we are interested in the expected value (mean) E of distribution D. Assuming the linearity of the expected value and a lack of correlation between $D_1, \dots, D_n$, we can derive the following equation:

Equation 5.1.2

$$E(D) = \frac{1}{m} \sum_{e=1}^{m} \sum_{v=1}^{n} E(D_{e,v})$$

After considering that $D_{e,v}$'s represent the expert estimations as triangular distributions in the form *Triang* ($a_{e,v}$, $b_{e.v}$, $c_{e,v}$), where a, b, and c are the known minimum, most likely, and maximum cost overhead, we obtain:

Equation 5.1.3 $$E(D) = \frac{1}{m}\sum_{e=1}^{m}\sum_{v=1}^{n}\frac{a_{e,v} + b_{e,v} + c_{e,v}}{3}$$

The derivation of an algorithm that implements Equation 5.1.3 is trivial. It delivers no error, because the results are calculated analytically. The *asymptotical runtime* of the analytical mean computation is **O(m·n)**, the respective *memory usage* is **O(1)**. In practice, computation on a 12x12 test matrix consists of 432 additions, which results in non-noticeable runtime of less than one millisecond on our test machine (desktop pc).

5.2 Distribution Computation

For AS2 and AS3, the density distribution *D* of the expert estimation matrix $D^{(m,n)}$ is needed. In order to analytically determine a function *D(x)* that describes the density distribution, again, we can use the mathematical representation introduced in Equation 5.1.1. The need density function of $D_v(x)$ can be computed using Equation 5.2.1, where $D_{e,v}(x)$ is the density function of the single triangular distribution.

Equation 5.2.1 $$D_v(x) = \frac{1}{m}\sum_{e=1}^{m} D_{e,v}(x)$$

But the next step would be the calculation of a sum of non-trivial probability functions (see Equation 5.1.1). The only analytical possibility to build a sum of non-trivial probability functions is to calculate their convolution [26]. In case of probability density function D, this means to solve an (n-1)-multiple integral equation (Equation 5.2.2).

Equation 5.2.2 $$D(x) = \int_0^x D_1(x - y_1)\int_0^{y_1} D_2(y_1 - y_2)...dy_1...dy_{n-1}$$

This, however, is not a trivial task, especially for larger *n*. In principle, the integral equation for given D_v's and x might be computed using the Fast Fourier Transformation [32] or multidimensional Gaussian quadratures [98]. Yet, both approaches deliver only approximations instead of analytically proper (exact) results.

So, we cannot present an exact analytical approach to determining the probability density function for expert estimation matrix $D^{(m,n)}$. Therefore, the next logical step is to take a look at Monte Carlo simulation and other competitive stochastic approaches.

6. STOCHASTIC APPROACHES

This section shows the adoption of two stochastic simulation approaches to the approximation of the cost overhead distribution as required in AS2 and AS3. The reason for choosing the first, standard Monte Carlo sampling (MC) is that it has been the approach applied in CoBRA® until now. In addition, it is the simplest simulation approach known and serves as a baseline for comparison with more sophisticated simulation approaches. The second approach, Latin Hypercube Sampling (LH), was chosen as best competitor based on the results of the literature research (see Section 3). We implemented LH in two variants with differences in runtime and required memory. In the final subsection, we explain why an experimental approach is needed to answer RQ2 and RQ3.

6.1 The Monte Carlo Approach

For AS2 and AS3, the density distribution D of the expert estimation matrix $D^{(m,n)}$ is needed. In order to obtain an approximation of D with the help of the standard MC, the algorithm follows the attempted sampling described in Section 2.2.1.

In the algorithm, we use the unbiased estimator as defined in Equation 6.1.1 to create samples of D (Equation 5.1.1).

Equation 6.1.1 $$D(\omega) = \sum_{v=1}^{n} \sum_{e=1}^{m} \chi_e(\xi^{(m)}(\omega)) D_{e,v}^{-1}(\xi_v^{(1)}(\omega))$$

$\xi^{(1)}{}_v(\omega)$ is an independent realization of the probability variable $\xi^{(1)}{}_v$ that is uniformly distributed on [0,1), $\xi^{(m)}(\omega)$ is a realization of $\xi^{(m)}$ that is uniformly distributed over the set {1,…,m}, χ_e is the *characteristic function* of *e* (Equation 6.1.2), and $D_{e,v}^{-1}$ is the inverse distribution function of the triangular distribution $D_{e,.v}$.

Equation 6.1.2 $$\chi_e(x) = \begin{Bmatrix} 1 \;\; if\; x = e \\ 0 \;\; else \end{Bmatrix}$$

This leads to the following algorithm in pseudo-code:

```
for i=0 to #Intervals : d[i] = 0;
for i=1 to z {
 sample = 0;
 for v=1 to n {
  sample +=
   D-
1[random({1,…,m})][v](random([0,1])
 }
 s = roundDown(sample);
 d[s] = d[s]+1;
}
for i=0 to #Intervals : d[i] = d[i] /
z;
return d;
```

Figure 6-1 MC sampling in pseudo-code

The *asymptotical runtime* for the MC sampling is **O(z·n)** for a $D^{(m,n)}$ matrix and z iterations[4] and the respective *memory usage* for the computation is **O(#intervals).**

6.2 The Latin Hypercube Approach

In order to obtain the density distribution D of the expert estimation matrix $D^{(m,n)}$ with the help of LH, we make use of sampling as described in Section 2.2.2. For the purpose of calculating D (see Equation 5.1.1) with the LH approach, we first need an unbiased estimator D(ω) of D. The starting point for the development of such an estimator is the realization of D(ω) presented for MC (Equation 6.1.1).

If we expect that the number of iterations (*z*), according to the number of strata is a multiplication of the number of experts (*m*), then we can optimize Equation 6.1.1 by replacing the random variable $\xi^{(m)}$ and the chi function by 1/m (Equation 6.2.1). This is possible because, when stratified by z, $\xi^{(m)}$ delivers (remember z is a multiplication of m) a sequence of exact values: (z/m · 1), (z/m · 2), …, (z/m · m).

[4] Under the assumption always valid in practice that m as well as the number of intervals is less than z.

Equation 6.2.1 $$D(\omega) = \sum_{v=1}^{n} \sum_{e=1}^{m} \frac{1}{m} D_{e,v}{}^{-1}(\xi_v^{(1)}(\omega))$$

The stratification of the remaining random variable $D_{e,v}$, which is required for implementing an LH algorithm (see Section 2.2.2), is quite simple because we can construct a random number generator with the help of the inversion method for $D_{e,v}$ (triangular distribution) [102]. This means that the only random variables that must be stratified are the $\xi^{(1)}{}_v$ ($v=1...n$), which are uniformly distributed on the range [0,1]. In practice, this is done by replacing $\xi^{(1)}{}_v$ by a random variable uniformly distributed between $[(\pi_v(i)-1)/z\ ,\ \pi_v(i)/z\ ]$:

Equation 6.2.2 $$\psi_v(\omega, i) = \frac{\xi_v^{(1)}(\omega) + \pi(i) - 1}{z}$$

where i is the current iteration, z the total number of iterations, and π_v represents random permutation over the elements $1...z$.

This way, we get the following estimator (Equation 6.2.3), which can be further optimized by removing $1/m$, the sum over m, and replacing e with $(i \bmod m) + 1$ (see Equation 6.2.4).

Equation 6.2.3 $$\xi(\omega) = \frac{1}{z} \sum_{i=1}^{z} \sum_{v=1}^{n} \sum_{e=1}^{m} \frac{1}{m} D_{e,v}^{-1} \frac{\xi_v^{(1)}(\omega) + \pi(i) - 1}{z}$$

Equation 6.2.4 $$\xi(\omega) = \frac{1}{z} \sum_{i=1}^{z} \sum_{v=1}^{n} D_{(i \bmod m)+1,v}^{-1} \frac{\xi_v^{(1)}(\omega) + \pi(i) - 1}{z}$$

The problem with implementing an estimator defined in such a way is that we need n independent permutations $p[v]$, where each $p[v][i]$ has the probability of $1/z$ of being equal to $c \in \{1,...,z\}$ (*).

We have implemented two algorithms that solve this problem in different ways. The first one, LH (Figure 6-2 (a)), randomly chooses c from the figures $1...z$ for each variable v in each iteration i. If the figure is flagged as having been chosen in an earlier iteration, a new figure is chosen; else the figure is used and flagged.

For a $D^{(m,n)}$ matrix and z iterations[5], the *asymptotical runtime* of the LH sampling algorithm is $\mathbf{O(z^2 \cdot n)}$, and *memory usage* for the computation is $\mathbf{O(z \cdot n)}$.

[5] On the basis of the assumption always valid in practice that m as well as *#intervals* are less than z^2.

```
for i=0 to #Intervals : d[i] = 0;
p[*][*] = 0;
for i=1 to z {
 sample = 0;
 for v=1 to n {
 c = random(0,…,z);
 while p[v][c] == 1 do p[v][c mod z
+1];
 p[v][c] == 1;
 sample +=
   D^-1[i mod m][v]( (random
([0,…,1])+c-1)/z )
   }
   s = roundDown(sample);
   d[s] = d[s]+1;
}
for i=0 to #Intervals : d[i] = d[i] /
z;
return d
```

(a)

```
for i=0 to #Intervals : d[i] = 0;
for v=1 to n {
 for i=1 to z {
    p[v][i] = i;
 }
 for i=1 to z {
   c = random(1,…,z);
   d = p[v][i];
   p[v][i] = p[v][c];
   p[v][c] = d;
 }
}
for i=1 to z {
 sample = 0;
 for v=1 to n {
   sample += D^-1[i mod m][v](
        (random ([0,1])+p[v][i]-1)/z )
 }
 s = roundDown(sample);
 d[s] = d[s]+1;
}
for i=0 to #Intervals : d[i] = d[i] /
z;
return d
```

(b)

Figure 6-2 sampling in pseudo-code: (a) LH, (b) LHRO

The second algorithm, LHRO (Figure 6-2 (b)), represents a runtime-optimized (RO) version. It creates all needed permutations *p[v]* before the sampling part of the algorithm starts and any iteration is executed. To do this, it permutes for each variable *v* all numbers from *1,...,z* at random and stores the result. The proof that the permutations obtained satisfy (*) is a simple induction over *z*.

Computational complexities of the LHRO for a $D^{(m,n)}$ matrix and z iterations[6] are *asymptotical runtime* of **O(z·n)**, and *memory usage* of **O(z·n).**

Please note that although both algorithms store O(z·n) values, their actual storage differs in that LH stores $z \cdot n$ flags (one bit each) and LHRO stores $z \cdot n$ integers (32 bits each).

6.3 Comparison of Stochastic Algorithms

When considering the MC and LH algorithms, we are unable to determine accuracy and efficiency regarding AS2 and AS3 based merely on theoretical considerations; neither can we say which one is more accurate or which one is more efficient in our context.

This is because we are not aware of any possibility to show that the errors of the algorithms based on the LH approach are lower for typical expert estimation matrices in CoBRA® application. Especially, we do not know, if the LH error is lower, to which extent it is lower compared with the error of MC.

[6] On the basis of the assumption always valid in practice that *m* as well as *#intervals* are less than *z*.

Additionally, if considering efficiency, for example as runtime efficiency, we have to execute the algorithms to measure the runtime. The runtime depends, among other things, on the chosen implementation language and execution environment and cannot be calculated theoretically only based merely on the pseudo-code representations.

But most notably, MC and LH are stochastic approaches, since their results suffer from a different degree of error, which not only depends on input data and number of iterations, but also on pure randomness. The same statement is true for the runtime. Therefore, in order to draw reliable conclusions regarding RQ2 and RQ3, we conduct an experiment where we compare the different approaches/algorithms in a typical CoBRA® application context.

7. EXPERIMENTAL STUDY

7.1 Experimental Planning

During experimental planning, we first define the constructs, accuracy, and efficiency we want to measure and use for comparing the sampling approaches. Next, we describe the independent and dependent variables relevant for our experiment and set up some hypotheses to answer RQ2. Then we present the experimental design we use to check our hypothesis and answer RQ2 and RQ3.

7.1.1 Construct: Accuracy

In order to quantify the accuracy and efficiency of the simulation techniques evaluated in the study, appropriate metrics had to be defined first. In this section, we start with the quantification of the accuracy construct. The next section deals with the accuracy-dependent efficiency construct.

Following the goal question metric paradigm [3], we had to consider the object of the study (i.e., the simulation methods), the purpose (i.e., characterization and comparison), the quality focus (accuracy), the perspective (CoBRA® method user) and the context. Whereas most of them are straightforward to define in our setting, we can see that two different *contexts* exist, which are described by application scenarios AS2 and AS3 (see Section 4). AS1 is not relevant at this point because it is covered by the analytical solution presented in the previous section.

But before we derived an accuracy measure for each of the two application scenarios, we had to solve a general problem in our setting: In order to measure the accuracy of the simulation output, we need the expected output to perform some kind of compaction between actual and expected output. However, the computation of an analytic correct output distribution is not possible for a complex distribution within an acceptable period of time (see Section 5.2). So instead, we have to use an approximation with a known maximal error regarding applied measures. In the following, when we speak of *reference data*, we mean the approximation of expected output based on the approximated expected distribution (called *reference distribution*). We obtain our reference distribution by generating distributions with MC and LH algorithms and a very large number of iterations in order to get extremely high precision. The distributions obtained in this way were next compared to each other with respect to the same measures that were used in the experiment. The magnitude of obtained errors determined the upper limit of accuracy that can be measured in the experiment. It means that a derivation below the identified limit is practically not possible to observe and thus considered not to be significant.

We introduced two major types of error measure to compare the results of the sampling algorithms considered: *Derivation over Cost Overhead* (DoCO) and *Derivation over Percentiles* (DoP).

The *DoCO* measure has been designed to calculate the deviation between the sampled distribution and the reference distribution with respect to AS1, where we ask how probable it is to be below a certain

cost (overhead) limit X, more formally: P(X≤CO). The DoCO value represents the derivation (absolute error) of P_S(X≤CO) calculated for the sampled distribution from the expected P(X≤CO), averaged over all meaningful CO values.

Equation 7.1.1 $$DoCO = \frac{1}{(MaxCO - MinCO)} \times \sum_{co=MinCO}^{MaxCO-1} \left| P_S(X < co) - P_R(X < co) \right|$$ where:

MaxCO = max {maximum cost overhead of D_S with probability greater than 0, maximum cost overhead of D_R with probability greater than 0}, and

MinCO = min {minimum cost overhead of D_S with probability greater than 0, minimum cost overhead of D_R with probability greater than 0}.

$P_S(X<co)$ returns the accumulated probability of the sampled distribution for all cost overhead values less than *co*, and

$P_R(X<co)$ returns the accumulated probability for the reference distribution for all cost overhead values less than *co*.

This measure is meant to give more importance to errors spread over a wider range of the distribution than to errors that have "local" impact on a narrow range of the distribution. This property of DoCO is important because narrow local errors appear less frequently when determining the probability for a given cost overhead interval. Consider, for example, the last two intervals in Figure 7-1. The deviations between sampled and reference distributions (error) in each of the two intervals have the same magnitude but opposite signs, so that they compensate each other (total error over both intervals is equal to zero). Therefore, considering any larger interval that contains both or none of them would not be affected by derivations (errors) measured separately for each interval. Furthermore, DoCO has the advantage of being independent of errors caused by the data model because it uses the same interval boundaries as the data model, whole numbers of cost overhead unit (short, %CO).

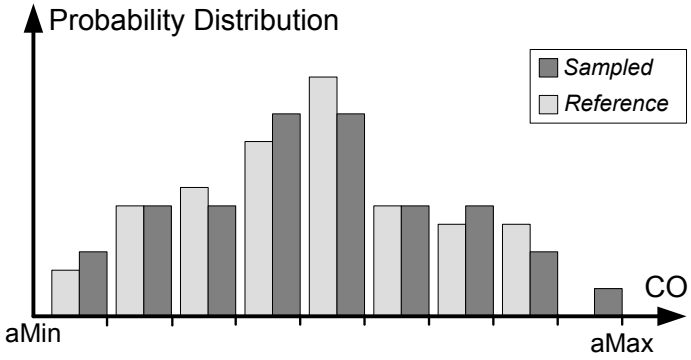

Figure 7-1: Probabilities of cost overhead values of sampled and reference distribution

The *DoP* (Derivation over Percentiles) measures were derived from the practice described in AS3: For instance, we might want to estimate the software cost (or cost overhead) that has a certain probability of not being exceeded. More formally, we search for the cost overhead (CO) value with a certain probability P(X≤CO) = p/100 associated to it, where p ∈ *N*, p ∈ [1, 99]. With DoP we capture the *average abso-*

lute error over all percentiles[7] (Equation 7.1.2) as well as the *maximum absolute error* that is reached in a single percentile (Equation 7.1.3):

Equation 7.1.2 $$DoP_{abs} = \frac{1}{99} \times \sum_{p=1}^{99} \left| CO_S(p) - CO_R(p) \right|$$

Equation 7.1.3 $$DoP_{\max} = \max \left\{ \bigcup_{p=1}^{99} \left| CO_S(p) - CO_R(p) \right| \right\}$$

where,

$CO_S(p)$ is the cost overhead value (CO_S) with $P_S(X \leq CO_S) = p/100$ and $CO_R(p)$ is the cost overhead value (CO_R) with $P_R(X \leq CO_R) = p/100$, where

$P_S(X \leq CO_S)$ is the accumulated probability of the sampled distribution for all cost overhead values less than or equal to CO_S, and

$P_R(X \leq CO_R)$ is the accumulated probability of the reference distribution for all cost overhead values less than or equal to CO_R.

The disadvantage of DoP is that the data model we used to store the sampling data[8] can theoretically cause an error of 0.5% in cost overhead (absolute error of 0.5). To explain this, let us consider two hypothetical distributions (Figure 7-2):

- A: $P_A(\alpha\text{-}1\ \%CO) = \infty$ ($\alpha \in N$) and $P_A(x) = 0$ for $x \neq (\alpha\text{-}1\ \%CO)$,
- B: $P_B(\alpha\text{-}0\ \%CO) = \infty$ and $P_B(x) = 0$ for $x \neq (\alpha\text{-}0\ \%CO)$.

In our data model, both distributions have the same representation: P([0, 1 %CO)) = 0, …, P([α-1 %CO, α %CO)) = 1, P([α %CO, α+1 %CO)) = 0, …

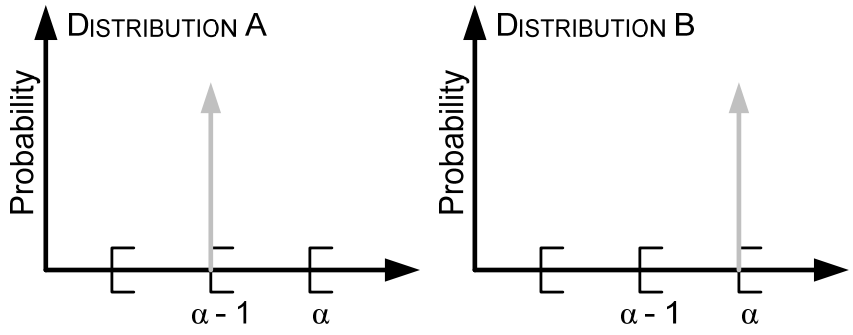

Figure 7-2: Example of two distributions with the same sampling data, but different Mean(D_R).

Using the data from the data model, we would calculate for both distributions $CO_A(1) = CO_B(1) = \ldots = CO_A(99) = CO_B(99)$, whereas in fact, distribution A should have $CO_A(1) = \ldots = CO_A(99) = \alpha$ -1 %CO

[7] The *Percentile* of a distribution of values is a number x_p such that a percentage p of the population values is less than or equal to x_p. For example, the 25th percentile of a variable is a value (x_p) such that 25% (p) of the values of the variable fall below that value.

[8] To store the results of the cost overhead distribution sampling, we used a data model in which every distribution interval is represented by a numeric value. Each value represents the probability that a cost overhead value contained in the respective interval occurs. Each of the half-opened intervals covers the same range: one unit of cost overhead (1%CO). Intervals with a probability value of zero are redundant and need not be stored in the model.

and B $CO_B(1) = \ldots = CO_B(99) = \alpha$ %CO. Therefore, in the worst case, the minimal error of the DoP measures that can be reached is 0.5 of the cost overhead unit (i.e., 0.5%CO).

With the understanding of derivation (error) being the opposite of accuracy, we define *accuracy* of the output as the inverse of the measured derivation regarding AS2: Accuracy = $(\text{DoCO})^{-1}$.

We used DoCO and not DoP to calculate accuracy because DoCO is, contrary to the DoP measures, independent of the imprecision of the sampling data model and therefore allows higher accuracy concerning the calculated derivation.

7.1.2 Construct: Efficiency

From the perspective of the CoBRA® method user, who wants to perform cost estimation or cost risk analysis, the accuracy of the result and the computation effort needed are the key factors that define efficiency. Therefore, following the goal/question/metric paradigm, *efficiency* is considered as accuracy per computation effort.

On the one hand, we can define *computation effort* as a measure of computational expense, measured by the number of executed iterations[9]; on the other hand, we can define computation effort as the execution time of the simulation algorithm in a typical execution environment, measured in milliseconds. Here, the number of iterations is independent of the development language and execution environment; measuring the execution time takes into consideration that the runtime of a single iteration can be different in different simulation approaches. Therefore, we defined two efficiency measures: *accuracy per iteration* (A/I) and *accuracy per millisecond* (A/ms).

Equation 7.1.4
$$A/I(D_S) = \frac{(DoCO_{abs})^{-1}}{\#Iterations}$$

Equation 7.1.5
$$A/ms(D_{abs}) = \frac{(DoCO_{abs})^{-1}}{execution\_time}$$

where *#Iterations* is the number of iterations performed and *excecution_time* is the runtime of the algorithm in milliseconds.

7.1.3 Independent and Dependent Variables

The variables that are controlled in this experiment (*independent variables*) are the chosen simulation algorithm, the number of iterations executed, the execution environment, as well as the simulation input (i.e., the matrix with the expert estimations). We control these variables, because we expect that they influence the measured *direct dependent variables* we are interested in: the simulation result (i.e., the sampled cost overhead distribution) and the execution time of the simulation algorithm. Based on the sampled distribution, we can calculate the derivation regarding the reference distribution (error) with the DoCO and DoP measures, and with our accuracy definition being based on DoCO. Knowing the accuracy, we can calculate efficiency regarding the number of iterations as well as regarding the required runtime. Therefore, our *indirect dependent variables* are derivation over cost overhead (DoCO), derivation over percentiles (DoP), accuracy, accuracy per iteration (A/I), and accuracy per millisecond (A/ms).

[9] The simulation consists of many recalculation cycles, so-called *iterations*. In each iteration, cost overhead values from triangular distributions across all cost drivers are sampled and used to calculate one possible total cost overhead value (according to the CoBRA® causal model).

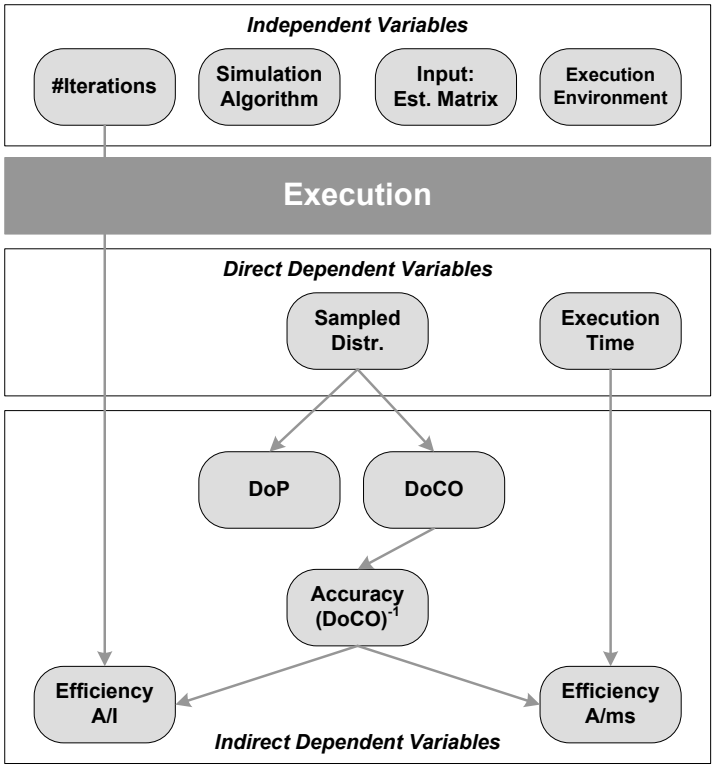

Figure 7-3: Relation between (direct/indirect) dependent and independent variables

7.1.4 Hypotheses

Based on the previous definition of variables and the research questions stated in Section 4, we have the following hypotheses and corresponding 0-hypotheses (which we want to reject):

H_A: The LH approach provides more accurate results (regarding DoCO) than the standard MC approach applied with the same number of iterations (≥10,000).

H_{A0}: The LH approach provides less or equally accurate results (regarding DoCO) than the standard MC approach applied with the same number of iterations (≥10,000).

H_B: The LH approach is more efficient (per iteration performed) than the standard MC approach when applied with the same number of iterations (≥10,000).

H_{B0}: The LH approach is less efficient than or as efficient as (per performed iteration) the standard MC approach when applied with the same number of iterations (≥10,000).

H_C: The LH approach is more efficient (per millisecond of runtime) than the standard MC approach when running in the same execution environment and with the same number of iterations (≥10,000).

H_{C0}: The LH approach is less efficient than or as efficient as (per millisecond of runtime) the standard MC approach when running in the same execution environment and with the same number of iterations (≥10,000).

There is a hypothesis that would be even more interesting to consider than H_C: *The LH approach is more efficient (per millisecond of runtime) than the standard MC approach when running in the same execution environment and produces a result of the same accuracy (regarding DoCO).* But this hy-

pothesis would be hard to investigate, since receiving enough LH and MC data points with the same result accuracy would be very difficult.

7.1.5 Experimental Design

The experimental design is based on the previously stated hypotheses and on the defined dependent and independent variables. In the following, subjects and objects involved in the experiment are presented; then, the chosen experiment design and statistical tests for checking the stated hypotheses are described.

Subjects: In our setting, the methods that should be compared to each other are performed by a computer since they are algorithms; consequently, there exist no subjects in the narrow sense. But following the statistical theory used in experimentation, the randomness (ω) required in the algorithms, visible as different instantiations of a random variable, and leading to different outcomes, can be seen as a 'perfect subject'. Perfect, because it is representative (perfectly random-sampled from the totality), available without any noticeable effort (simply generated by a computer as a set of pseudo-random numbers), and really independent of any other 'subject'.

Object: In order to get a representative study object for the context of cost estimation (i.e., realistic simulation algorithms input), we decided to use data from a real CoBRA® project. For that purpose, we employed data gained during the most recent industrial application of the CoBRA® method, which took place in the context of a large Japanese software organization [98]. The input data consisted of measurement data (size, effort) and experts' assessments (cost drivers, effort multipliers) collected across 16 software projects. The resulting simulation input was a 12x12-triangular distribution matrix, where the 144 triangular distributions represent estimates provided by twelve experts for twelve cost drivers. This matrix size is at the upper level when compared to other CoBRA® projects known to the authors.

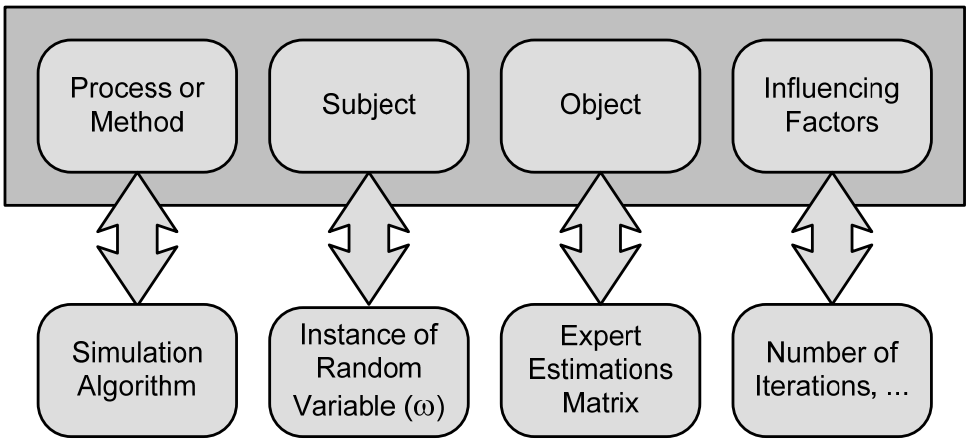

Figure 7-4: Mapping of the terms: method, subject, object, and influencing factors for our experiment setting

Since the computer can generate without noticeable effort any number of pseudo-random numbers, we have as many 'subjects' as we need. Therefore, we decided to conduct a *multi-test within object study* (i.e., we have multiple subjects, but only one object), where each subject (as a concrete instantiation of used random variables) is used only in one test, which means we have a *fully randomized design*. The factor whose influence should be observed is the chosen simulation algorithm with the *three treatments*: standard Monte Carlo (MC), Latin Hypercube (LH), and Latin Hypercube Runtime Optimized (LHRO). Since the number of iterations influenced the measured dependent variables accuracy and execution time, we chose a *block design* with 6 blocks with respect to the number of iterations performed: 10k,

40k, 160k, 640k, 2.56M, and 10.24M. Per treatment and block, 3 tests were performed and the median regarding the used derivation measure was selected. This was done to avoid outliers with respect to measured dependent variables (DoCO, DoP); the median was chosen because it is more robust than the mean.

	Simulation Algorithms		
Iterations	**MC**	**LH**	**LHRO**
10,240,000	1,2,3	4,5,6	7,8,9
2,560,000	10,11,12	13,14,15	16,17,18
640,000	19,20,21	22,23,24	25,26,27
160,000	28,29,30	31,32,33	34,35,36
40,000	37,38,39	40,41,42	43,44,45
10,000	46,47,48	49,50,51	52,53,54

Table 7-1 Block design: Distribution of tests over simulation algorithm and number of iterations

To confirm our hypothesis, we conducted one-sided *Sign Tests* between the MC and both LH approaches for each of the presented hypotheses. In addition, we performed one-sided *paired t-Tests* on the normalized[10] DOCO values (i.e., inverse accuracy) to provide evidence of the accuracy improvement by LH (H_A). The stricter t-Test is possible at this point, since we consider simulation errors (which can be assumed to follow a normal distribution).

7.2 Experimental Operation

The experiment was conducted as prescribed in the experiment design. The investigated simulation algorithms were implemented in Java within the existing CoBRA® software tool. Tests were performed on a Desktop PC with Windows XP (SP2), AMD AthlonXP3000+ and 1GB DDR-RAM.

For each trail, we let our test system compute the approximated distributions and stored them for further analyses. In addition, the simulation runtime in milliseconds was kept. *Problems* that occurred during operation were that tests 4 to 6 could not be performed due to extreme computation time (we stopped after one hour) and tests 7 to 9 as well as tests 16 to 18 were not performable due to stack overflows (memory problem).

The resulting distributions were used to calculate the corresponding DOCO and DOP values, ending up with the following tuple for each block of our test design (triple of tests):

- Median DoCO, corresponding simulation runtime
- Median DoP_{abs}, corresponding simulation runtime
- Median DoP_{max}, corresponding simulation runtime

The first kind of tuple was then used to get the needed accuracy (A) and efficiency values (A/I, A/ms) for each block.

[10] DOCO values are normalized by multiplying them with the square of iterations performed (i.e., the expected reduction of derivation).

7.3 Experimental Results

In this section, we present and discuss the results of our experiment. We check the stated hypotheses and compare the simulation approaches regarding accuracy and efficiency. The algorithms considered are the previously introduced standard Monte Carlo sampling (MC), Latin Hypercube sampling (LH), and the runtime optimized version of Latin Hypercube sampling (LHPO).

7.3.1 Accuracy of Simulation Algorithms

First, we look at the accuracy of the simulation approaches, more precisely at the derivation over cost overhead (DoCO) as inverse accuracy. In addition, we preset the derivation over percentiles results (DoP). Even if they are not used to calculate and compare the accuracy of the algorithms, they give some hints about the magnitude of the expected error when applying the different approaches.

	DoCO		
Iterations	**MC**	**LH**	**LHRO**
10,240,000	3.69E-05	-	-
2,560,000	8.13E-05	5.50E-05	-
640,000	2.19E-04	1.36E-04	1.75E-04
160,000	4.17E-04	2.66E-04	1.81E-04
40,000	8.61E-04	4.97E-04	4.92E-04
10,000	1.77E-03	1.11E-03	1.42E-03

Table 7-2 Comparison of derivation over cost overhead

	DoP average		
Iterations	**MC**	**LH**	**LHRO**
10,240,000	2.34E-02	-	-
2,560,000	3.85E-02	3.31E-02	-
640,000	2.84E-01	9.42E-02	1.04E-01
160,000	1.49E-01	2.07E-01	1.03E-01
40,000	6.55E-01	2.34E-01	2.93E-01
10,000	6.73E-01	5.62E-01	8.82E-01

Table 7-3 Comparison of average derivation over percentiles

	DoP max		
Iterations	**MC**	**LH**	**LHRO**
10,240,000	7.09E-02	-	-
2,560,000	4.03E-01	2.15E-01	-
640,000	4.90E-01	7.09E-01	8.56E-01
160,000	5.21E-01	6.28E-01	5.19E-01

40,000	1.22E+00	1.95E+00	2.28E+00
10,000	4.09E+00	2.36E+00	4.28E+00

Table 7-4 Comparison of maximum derivation over percentiles

Discussion: Considering Table 7-2, one can see that both the LH and the LHRO algorithms have comparable accuracy (regarding DoCO) for a given number of iterations. This is not surprising because both are based on the general LH sampling approach and both differ only with respect to the way they were implemented. The LH algorithms outperform the MC algorithm in *accuracy* in any given number of iterations. The relative accuracy gain with respect to MC varies for LH (mean 0.60) and LHPO (mean 0.63) around 0.6; this means a 60% improvement of accuracy of LH approaches over MC. The number of iterations seems to have no noticeable influence on this.

Hypothesis testing: We check our hypothesis H_A with a sign test and can reject the null hypothesis at an α level of .05 for LH; performing a paired student's t-test on normalized DoCO values, we can reject the null hypothesis for both implementations, LH and LHRO. Therefore, we can conclude that the LH approach delivers better results than the standard MC approach in the context of our application scenario and the range of 10,000 to 2,560,000 iterations.

Hypothesis H_A	LH	LHRO
Sign test	**p = 0.031**	p = 0.063
Paired Student's t-test	**p > 0.001**	**p = 0.027**

Table 7-5 Test results regarding hypothesis H_A

7.3.2 Efficiency of Simulation Algorithms

After we considered the accuracy of the different algorithms, we next looked at their efficiency.

	A/I			A/ms		
Iterations	**MC**	**LH**	**LHRO**	**MC**	**LH**	**LHRO**
10,240,000	2.65E-03	-	-	5.50E-01	-	-
2,560,000	4.80E-03	7.10E-03	-	9.99E-01	3.48E-02	-
640,000	7.14E-03	1.15E-02	8.94E-03	1.46E+00	1.06E-01	1.07E+00
160,000	1.50E-02	2.35E-02	3.45E-02	3.12E+00	3.86E-01	4.17E+00
40,000	2.90E-02	5.03E-02	5.08E-02	6.21E+00	1.03E+00	4.20E+00
10,000	5.66E-02	9.02E-02	7.03E-02	1.21E+01	4.44E+00	1.50E+01

Table 7-6 Comparison accuracy: per iteration (A/I) and per millisecond (A/ms)

Efficiency is measured as accuracy per iteration as well as accuracy per millisecond of runtime (Table 7-6). In the following, we discuss the results obtained and check our efficiency-related hypotheses.

Discussion: The efficiency index *accuracy per iteration* shows that the LH algorithms outperform the MC algorithm. The relative accuracy per iteration gain shows the same picture as the accuracy improvement calculated for LH (mean 0.60) and LHPO (mean 0.63). This means around 60% improved A/I of the LH approaches over MC. An interesting aspect, which we can observe in Table 7-6, is that I/A decreases at the factor 2 when the number of iterations is quadrupled by any of the three sampling

approaches. Thus, it can be concluded that the DoCO error decreases for all algorithms at the rate of $\Theta(\#\text{iterations}^{-1/2})$, confirming our expectation based on our theoretical results (Equation 7.4.10).

The picture changes when looking at Table 7-6, which presents the *accuracy per millisecond* results (A/ms)). The accuracy-per-iteration advantage of the LH approaches is leveled down by the additional effort for creating the needed random and independent permutations. The precision per millisecond values of the MC and the LHRO algorithms at a given number of iterations are comparable, so that the performance of both algorithms for a given number of iterations can be considered as being nearly equivalent. Both LHRO and MC outperform the LH algorithm that uses (in contrast to the LHRO) a runtime-consuming (yet memory-saving) method to create the permutations. The higher the number of iterations, the higher the gap between the A/ms value of the LH algorithm and the other algorithms. For 10,000 iterations, the A/ms efficiency of the LH algorithm is 2.7 times worse; for 2,560,000 iterations, the efficiency is worse by the factor 28.

Our conclusion that the A/ms performance of the MC and LHRO algorithm is nearly equivalent is true only if comparing them with regard to the same number of iterations. Nevertheless, for the same number of iterations, the LHRO algorithm delivers a higher accuracy than the MC algorithm. For example, the accuracy of the LHRO algorithm with 160,000 iterations is nearly equal to the accuracy of the MC algorithm with 640,000 iterations, but when comparing the corresponding A/ms values, the efficiency of the LHRO algorithm is almost twice as good. The problem is that a statistical test to prove this conclusion cannot be provided because we would need comparison data with equal accuracy for all algorithms and such data are difficult to obtain.

In order to perform a nondiscriminatory comparison between the performance of the MC and LHPO approach with respect to runtime in ms (A/ms), they have to be compared on the same level of accuracy. Because we do not have the data for MC and LHPO on the same accuracy level, the relation between the number of iterations and the achieved accuracy is considered (Figure 7-5). For MC, there seems to be a linear relationship between the square root of the number of iterations and the achieved accuracy. A linear regression through the origin results in the following equation with an r^2 value of more than 0.999:

Equation 7.3.1 $$A_{MC} = 5.7658 \cdot \sqrt{z}$$

Using Equation 7.1.3, the number of iterations (z) that MC requires to reach the accuracy of the LHPO executions at 640,000, 160,000, 40,000, and 10,000 iterations can be calculated. The results are 982,189, 918,150, 124,263, and 14,917 iterations. Additionally, a linear relationship between the number of iterations and the runtime of the implemented algorithm can be recognized for the implemented MC algorithm (r^2>0,999). Therefore, the runtime of MC can be calculated for 982,189, 918,150, 124,263, and 14,917 iterations (i.e., for the number of iterations MC requires to reach the same accuracy as LHPO at 640,000, 160,000, 40,000, and 10,000 iterations). Comparing these runtimes with corresponding runtimes of LHPO shows an average reduction of runtime through LHPO of ~30%, equivalent to the efficiency (A/ms) improvement of ~77% (mean).

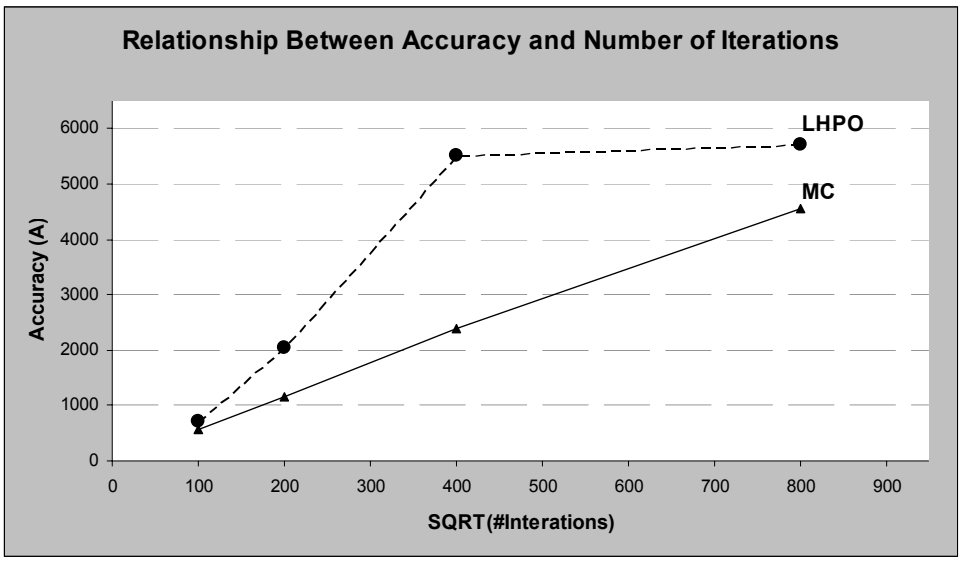

Figure 7-5: Relationship between Accuracy and Number of Iterations

Hypothesis testing: We checked our hypotheses H_B and H_C with a sign test at an α level of .05, but we could only reject H_{B0} for LH. Therefore, based on the limited number of data points, we can only conclude that the LH approach is more efficient regarding the number of iterations (A/I).

Hypothesis H_B	**LH**	**LHRO**
Sign test	**p = 0.031**	p = 0.063
Hypothesis H_C	**LH**	**LHRO**
Sign test	p = 0.969	p = 0.500

Table 7-7 Test results regarding hypotheses H_B and H_C

7.4 Validity Discussion

First, we discuss typical threats to validity that are relevant for our experiment. Next, we present upper bound estimation for average simulation error with respect to application scenario AS2 (DoCO) and the dependent accuracy measure.

7.4.1 Threats to Validity

In this section, we consider possible threats to validity based on the experiment design and its operation. We present them in the categories proposed by Wohlin et al. [106]: First, conclusion validity, which concerns the problem of drawing a wrong conclusion from the experiment outcome; then we look at internal validity, which is threatened by uncontrolled influences distorting the results. Next, construct validity is addressed ("Do we measure what we propose to measure?"), and finally, external validity is regarded, which describes how well results can be generalized beyond our experimental setting.

Conclusion validity:

- The sign test, which is primarily used in our experiment, is one of the most robust tests. It makes no assumption beyond those stated in H_0. The paired student t-test is only used when comparing normalized simulation errors that can be assumed to be normally distributed.
- One problem is the low statistical power of the sign tests that results from the low number of pairs that could be compared. When testing LHRO against MC, for example, we could compare only four instead of the expected six pairs, since we were not able to execute tests 7, 8, 9, 16, 17, and 18, which result in stack overflows due to their memory requirements.

Internal validity:

- The chosen implementation in Java and the execution environment influence the runtime of the algorithms. Therefore, other implementations executed in different environments can result in different runtime and thus efficiency. To keep the results of different algorithms and iteration numbers comparable, all algorithms are implemented in the same programming language and executed on the same system. But we cannot assure that each algorithm is implemented in the most runtime-efficient way possible.
- The measures derived for AS3 (DoP) suffer from measurement inaccuracy as a result of the data model. Therefore, the algorithms are compared based on the measure derived for AS2 (DoCO), which is independent of this bias.
- The used stochastic simulation algorithms have no deterministic result. Therefore, there is a minor but not dismissible possibility of getting a very unusual result if executing a simulation. In order to avoid the presence of such outliers in analyzed data, we execute three trails (i.e., tests) for each setting and take the median.
- The reference distribution applied to measure the accuracy of the simulation results is not the analytically correct output distribution. Instead, we have to use an approximation with a known error regarding applied measures. The reason is that computation of an analytically correct output distribution is not possible for a complex distribution within an acceptable period of time. For a more detailed discussion, see Section 7.1.

Construct validity:

- The chosen constructs that represent accuracy and efficiency are selected carefully and based on the understanding of accuracy and efficiency in typical application scenarios of cost estimation and cost risk analysis. But we provide no empirical evidence that this is true for the majority of users.

External validity:

- The number of experts and variables as well as the characteristics of input distributions can influence the accuracy and performance of the algorithm. These input data (represented by the distribution matrix) are different in different CoBRA® application scenarios. We performed the experiment only for one set of input data; therefore, generalizability of results (i.e., representativity of input data) is an issue. To reduce this threat as much as possible, input data was taken from an actual industrial application. The input data have a complexity in the number of experts and variables at an upper, but not untypical, level compared with other known applications.

7.4.2 Analytical Considerations

As previously mentioned, based on theoretical considerations, we cannot decide whether LH is more accurate and efficient than MC in our context. But we can give a lower boundary for the expected accuracy (regarding DoCO) that is true for both MC and LH. Such a lower boundary for accuracy allows us, on the one hand, to perform a simple validity check on the measured accuracy values; on the other hand, it can increase the external validity of our experiment results.

To derive this lower boundary, we first consider the MC algorithm. When we look at a single *individual interval* I_s, it appears that for this interval, the algorithm does nothing else than estimate the result of the integral of the density function D(x) over this interval (Equation 7.4.1) with the standard MC integration [51].

Equation 7.4.1 $$I_s = \int_s^{s+1} D(x)\,dx$$

Since interval size is one (measured in percentage of cost overhead), I_s delivers the average density of D on the interval [s,s+1]. This is interesting because the error of the whole distribution can now be expressed with the help of the errors of the I_s estimators η. Considering our algorithm (Figure 6-1), the estimation of I_s is calculated as follows:

Equation 7.4.2 $$E_s(z) = \frac{1}{z}\sum_{i=1}^{z}\eta_i, with\ \eta_i = \chi_{[s,s+1]}(D(\omega_i))$$

$D(\omega_i)$ is the i-th independent realization of the random variable $D(\omega)$ with distribution D and $\chi_{[s,s+1]}$ is the characteristic function for the interval [s,s+1] (Equation 7.4.3).

Equation 7.4.3 $$\chi_{[s,s+1]}(x) = \begin{cases} 1 & if\ x \in [s, s+1] \\ 0 & else \end{cases}$$

This estimator is *unbiased*, meaning $\lim_{z\to\infty} E_s(z) = I_s$. This is clear when we recall that the expected value is the ratio of $| D_S |$ to $| D |$, and that $| D | = 1$ (D is a probability density distribution).

Now, in order to calculate the *expected error* for *z* iterations with Equation 2.2.2, we need the standard deviation (or variance) of our estimator. Having the variance definition (Equation 7.4.4) and $E(\eta) = E(\eta^2) = I_s$, we derive upper bounds for estimator variance (Equation 7.4.5).

Equation 7.4.4 $$V(\eta) = E(\eta^2) - E(\eta)^2$$

Equation 7.4.5 $$V(\eta) = I_s - I_s^2 \le I_s \le 1$$

The *expected error* for *z* iterations can be calculated with Equation 2.2.2 as follows (Equation 7.4.6):

Equation 7.4.6 $$EAR_{I_s} \le \frac{\sqrt{I_s}}{\sqrt{z}}$$

Example: If we have an average value of 0.01 over the interval [s,s+1] and run 10,000 iterations, we would get an EAR_{Is} of 0.1/100 = 0.001 (corresponding to a relative error of 10%).

In addition, Equation 7.4.6 can be used to provide a theoretical upper boundary estimation of *Deviation over Cost Overhead* (DoCO) measure (Equation 7.1.1). The term $\left| P_S(X < i) - P_R(X < i) \right|$

in the DoCO measure represents nothing else than the absolute derivation of the integral of distribution D over the interval [0,i] (I_i). Expressing I_i with the intervals over [s,s+1] (Equation 7.4.7) and reusing Equation 7.4.6 allows us to obtain a final estimation of the expected error estimating I_i and DoCO value (Equation 7.4.8 and Equation 7.4.9 respectively), when executing *z* iterations.

Equation 7.4.7
$$I_i = \sum_{s=0}^{i-1} \int_{s}^{s+1} D(x)\,dx = \sum_{s=0}^{i-1} I_s$$

Equation 7.4.8
$$EAR_{Ii} \leq \frac{\sqrt{\sum_{s=0}^{i-1} I_s}}{\sqrt{z}}$$

Equation 7.4.9
$$DoCO \leq \frac{1}{\sqrt{z} \cdot \#Intervals} \times \sum_{i=1}^{\#Intervals} \sqrt{\sum_{s=0}^{i-1} I_s}$$

At a given number of iterations and intervals, DoCO becomes maximal if the double sum in Equation 7.4.9 becomes maximal. This, in turn, happens when I_1 is 1 and all other I_s are equal to 0 (consider that the sum of all I_s = | D | = 1).

Equation 7.4.10
$$DoCO \leq \frac{1}{\sqrt{z}} \quad and \quad Accuracy \geq \sqrt{z}$$

Equation 7.4.10 is a weak assessment, but it is sufficient for our purpose. A stronger assessment (dependent on #Intervals) is possible regarding the exact variance of I_s.

Example: If we sample a distribution D with MC and 10,000 iterations, we get an expected DoCO value of less than 0.01.

The expected error of the LH approach can be estimated with the same equations as those used by the standard MC approach. In particular, Equation 7.4.10 holds for the expected DoCO value. The reason is that the LH approach is a special case of the standard MC approach and converges at least as fast as the MC approach (see Section 2.2.2).

8. SUMMARY

In this chapter, we investigated the potential impact of simulation techniques on the estimation error of CoBRA®, as an exemplary cost estimation and cost risk analyzing method. This was done by deriving an analytical solution for point estimations and performing an experiment with industrial project data [98]. This experiment was necessary to compare the stochastic simulation approaches when applied to the sophisticated analysis of project costs. We could answer the question of whether there exist techniques that deliver more accurate results than simple Monte Carlo sampling or performing the simulation in a more efficient way. In addition, we present results regarding the magnitude of accuracy and efficiency improvement that can be reached through the use of Latin Hypercube as an alternative sampling technique. The summarizing presentation of results is structured according to the three detailed research questions (RQ) defined in Section 4 and the context of the three different application scenarios (AS); see Table 8-1.

With respect to RQ1, we showed in Section 5 that analytical mean computation (AS1) is possible and should be preferred over computation based on simulation. First of all, in contrast to stochastic ap-

proaches, the analytical approach provided repeatable and error-free results. Besides that, it has an extremely short runtime (<1ms), i.e., it has very low application costs, which makes the approach highly efficient. In that context (analytical approach is feasible and has excellent runtime), considering research questions RQ2 and RQ3 made no sense. Based on these results, the analytical computation of mean cost overhead should be employed within the CoBRA® method (and implemented by a tool supporting the method). Yet, we could not find a feasible analytical approach to compute the cost overhead distribution required in application scenarios AS2 and AS3. Thus, we had to sample the distribution with selected stochastic methods.

Application Scenario	**RQ1** Analytical feasibility	**RQ2** Comparison: Accuracy & Efficiency	**RQ3** Magnitude of Improvement
AS1	**Yes**	*Not relevant*	*Not relevant*
AS2	No	**H_{A0} rejected** (accuracy A)	**~ 60% accuracy (A)** and **~ 77% efficiency (A/ms)** gain by LHPO
		H_{B0} rejected (efficiency A/I)	
		H_{C0} <u>not</u> rejected (efficiency A/ms)	
AS3	No	-	-

Table 8-1 Answered research questions

Since the measure derived to calculate the sampling error regarding the needs of AS3 suffers from inaccuracy based on the sampling data model, we calculated accuracy and efficiency only based on the error measure derived for AS2 (namely, DoCO). Therefore, we can answer RQ2 and RQ3 with certainty only for AS2.

Regarding RQ2, we can say that the Latin Hypercube (LH) algorithm and its performance-optimized version (LHRO) provide comparable accuracy for a given number of iterations. This is not surprising, since both are based on the same general sampling approach and practically differ only with respect to the way they were implemented. Yet, they outperform the Monte Carlo (MC) algorithm regarding *accuracy* at any number of iterations, which we proved with a sign test at α level .05 (H_A).

The LH algorithms present about 60% improvement in accuracy over MC on average, with respect to DoCO (RQ3). Moreover, the number of iterations does not seem to have a noticeable influence on the improvement rate.

In case of LH, however, high accuracy is achieved at the expense of lower performance (increased runtime) as compared to MC and LHRO. Already at 10,000 iterations, efficiency measured as accuracy per millisecond (A/ms) of LH differs by a factor of 2.7 and decreases with an increasing number of iterations. It seems that LHRO has a better efficiency regarding runtime (A/ms) than MC at a given level of accuracy (~77% estimated, based on 8 data points), but we could not prove this statistically, since the required data are difficult to obtain. The data for comparing the approaches at a given number of simulation runs are easier to obtain. However, the magnitude of accuracy improvement decreases with increasing accuracy, so an improvement at a given number of simulation runs could not be proven (H_C).

With respect to the number of simulation runs (i.e., iterations), LH has a better efficiency than MC, which could be proven by rejecting the corresponding null hypothesis (H_{B0}).

Summarizing, since the results of our experiment showed that LHRO has a statistically significant higher accuracy than the MC when employed within CoBRA® and seems to also have a higher efficiency regarding runtime compared to MC at the same level of accuracy, LHRO should be the preferred simulation technique to be employed within the CoBRA® method. We showed that with 640,000 iterations on the test PC in less than six seconds, the LHRO algorithm delivers more than sufficiently accurate results.

It is, however, necessary to stress that a simulation algorithm is only responsible for transforming CoBRA® input data (expert estimations) into model output (cost overhead distribution) and is practically independent of the quality of the input data. Although LHRO significantly improves sampling accuracy (60% increase) and efficiency (77% increase) compared to MC, its impact on the overall output of estimation strongly depends on the characteristics of the specific estimation method. For instance, the more sampling runs (e.g., for numerous input factors) are performed, the larger the gain in computational performance of the whole estimation method. Regarding the estimation accuracy, the error introduced by even the worst sampling algorithms considered here (<4.3%) is only a small fraction of the potential error caused by, for example, low quality of the input data, which in case of the CoBRA® method may easily exceed 100% [98]. Yet, considering that for valid data, CoBRA® is capable of providing estimates with less than 15% inaccuracy [98], the error introduced by the simulation technique alone may be considered significant. Therefore, we recommend LHRO as the preferable simulation technique for the CoBRA® method. Compared to MC, LHRO does not entail significant implementation overhead but may provide significant estimation precision and efficiency gains.

Based on the presented theoretical error estimations (especially Equation 7.4.6 and Equation 7.4.10) and confirmed by the experiment results, which reflect a typical application scenario of simulation for cost estimation purposes, the aforementioned conclusions might, in practice, be generalized on any cost estimation method employing simulation. Specific gains in accuracy and performance depend, however, on the number of simulation runs and operations performed on the sampling output (e.g., multiplying sampling outputs may multiply overall error). Moreover, the contribution of the simulation error to the overall estimation inaccuracy depends on the magnitude of error originating from other sources such as invalid estimation (and thus sampling) inputs. In practice, for instance, the quality and adequacy of input data proved, over the years and over various estimation methods, to have a large impact on prediction quality [86]. Therefore, one of the essential method acceptance criteria should be the extent to which a certain estimation method copes with potentially sparse and messy data.

Summarizing, selecting the LHRO simulation technique instead of the traditionally acknowledged MC may provide non-ignorable effort estimation accuracy and performance gains at no additional overhead. The relative estimation improvement depends, however, on the magnitude of other estimation errors. In practice, inadequate, redundant, incomplete, and inconsistent estimation inputs are still responsible for most of the loss in estimation performance (accuracy, precision, efficiency, etc.) Therefore, one of the essential method acceptance criteria should be the extent to which a certain estimation method copes with potentially sparse and messy data. In addition to efforts spent on improving the quality of data (both measurements and experts' assessments), future research should focus on a method that can handle low-quality data and support software practitioners in building appropriate, goal-oriented measurement programs. One direction of work might be, as already proposed in [98], a framework that combines expert- and data-based approaches to iteratively build a transparent effort model. The framework proposes additional quantitative methods to support experts in achieving more accurate assessments as well as to validate available measurement input data and underlying processes. Successful results of the initial validation of the proposed approach presented in [98] are very promising.

ACKNOWLEDGEMENTS

We would like to thank OKI for providing the CoBRA® field data used to perform the study, the Japanese Information-technology Promotion Agency (IPA) for their support, Prof. Dr. Dieter Rombach and Marcus Ciolkowski from the Fraunhofer Institute for Experimental Software Engineering (IESE) for their valuable comments to this chapter, and Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of the chapter.

REFERENCES

[1] Ahmed M.A., Saliu M.O., AlGhamdi J., "Adaptive fuzzy logic-based framework for software development effort prediction." Information and Software Technology, 47(1):31-48, January 2005.

[2] Aguilar-Ruiz J.S., Ramos I., Riquelme J.C., Toro M., "An Evolutionary approach to estimation software development projects." Information and Software Technology, 43:875-882, 2001.

[3] Angelis, L., Stamelos, I., Morisio, M., "Building a software cost estimation model based on categorical data." Proceedings of the IEEE 7th International Symposium on Software Metrics. England, UK, pp. 4-15, 2001.

[4] Avramidis A. N., Wilson J. R., *Correlation-Induction Techniques for Estimating Quantiles in Simulation Experiments*. Proceedings of The Winter Simulation Conference, IEEE Press, 1995.

[5] Avramidis A. N., Wilson J. R., *Correlation-Induction Techniques for Estimating Quantiles in Simulation Experiments*. Technical Report 95-05, Department of Industrial Engineering, North Carolona State University, Raleigh, North Carolina.

[6] Basili V. R., Rombach H. D., *The TAME project: Towards improvement–oriented software environments*. IEEE Transactions on Software Engineering, 14(6):758–773, June 1988.

[7] Beck K., Fowler M., Planning Extreme Programming, Addison-Wesley, October 2000.

[8] Boehm B.W., Abts C., Brown A.W., Chulani S., Clark B.K., Horowitz E., Madachy R., Refer D., Steece B., Software Cost Estimation with COCOMO II. Prentice Hall, 2000.

[9] Boehm B., Abts C. Chulani S., "Software development cost estimation approaches - A survey." Annals of Software Engineering, 10:177–205, Springer Netherlands, 2000.

[10] Boehm B.W., Software Engineering Economics. Prentice-Hall, 1981.

[11] Breiman L., Friedman J., Ohlsen R., Stone C. Classification and Regression Trees. Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[12] Briand, L.C., Wieczorek, I. Software Resource Estimation; in: Marciniak J.J. (ed.), *Encyclopedia of Software Engineering*, 2:1160-1196, John Wiley & Sons, 2002.

[13] Briand L., Langley T., Wieczorek I., "A replicated Assessment and Comparison of Common Software Cost Modeling Techniques." Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.

[14] Briand, L.C., El Emam, K., Bomarius, F. CoBRA: A Hybrid Method for Software Cost Estimation, Benchmarking and Risk Assessment, *Proceedings of the 20th International Conference on Software Engineering*, 1998, pp. 390-399.

[15] Briand L. C., Basili V. R., Thomas W. M. "A Pattern Recognition Approach for Software Engineering Data Analysis." IEEE Transactions on Software Engineering, 18(11):931-942, November 1992.

[16] Burgess C.J., Lefley M., "Can genetic programming improve software effort estimation? A comparative evaluation". Information and Software Technology, 43(14):813–920, 2001.

[17] Calzolari F., Tonella P., Antoniol G., "Maintenance and testing effort modeled by linear and nonlinear dynamic systems." Information and Software Technology, 43(8):477-486, July 2001.

[18] Chulani S., Boehm B.W., Steece B. "Bayesian Analysis of Empirical Software Engineering Cost Models." IEEE Transactions on Software Engineering, 25(4):573-583, July/August 1999.

[19] Cuelenaere A.M.E., van Genuchten M.J.I.M., Heemstra F.J., "Calibrating Software Cost Estimation Model: Why and How," Information and Software Technology, 29(10): 558-567, December 1987.

[20] De Lucia A., Pompella E., Stefanucci S. "Assessing effort estimation models for corrective maintenance through empirical studies." Information and Software Technology, 47:3-15, 2004.

[21] Dolado J.J., "On the problem of the software cost function." Information and Software Technology, 43:61-72, 2001.

[22] Elkjaer M. "Stochastic Budget Simulation." International Journal of Project Management, 18(2):139-147, April 2000.

[23] Feller W., *An Introduction to Probability – Theory and Its Application*. John Wiley & Sons. 1971.

[24] Fenton N. Marsh W. Neil M. Cates P. Forey S. Tailor M. "Making resource decisions for software projects." Proceedings of the 26th International Conference on Software Engineering, May 2004, pp. 397- 406.

[25] Ferens D.V., Christensen D.S.: "Does Calibration Improve Predictive Accuracy?" Cross-Talk: The Journal of Defense Software Engineering, April 2000, pp. 14-17.

[26] Feller W., *An Introduction to Probability – Theory and Its Application*. John Wiley & Sons. 1971.

[27] Garratt P.W., Hodgkinson A.C. "A Neurofuzzy Cost Estimator." Proceedings of the 3rd International Conference Software Engineering and Applications, Arizona, USA, October 1999, pp. 401-406.

[28] Gartner Inc. press releases, *Gartner Survey of 1,300 CIOs Shows IT Budgets to Increase by 2.5 Percent in 2005*, 14th January 2005, http://www.gartner.com/press_releases/pr2005.html.

[29] Gartner Inc. press releases, *Gartner Says Worldwide IT Services Revenue Grew 6.7 Percent in 2004*, 8th February 2005, http://www.gartner.com/press_releases/pr2005.html.

[30] Gray A., MacDonell S,. "Applications of Fuzzy Logic to Software Metric Models for Development Effort Estimation." Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society. Syracuse NY, USA, 1997, pp. 394-399.

[31] Guo-Tong, Li-Aiqun, Miao-Changqing, *Monte Carlo numerical simulation and its application in probability analysis of long span bridge*. Journal of Southeast University (English Edition), * 21(4):469-473, China, December 2005.

[32] Hayashi I., Iwatsuki N., *Universal FFT and its application*, Journal of the Japan Society of Precision Engineering, 25(1):70-75, 1990.

[33] Heiat A., "Comparison of artificial neural network and regression models for estimating software development effort." Information and Software Technology, 44(15):911-922, December 2002.

[34] Helton J. C., Davis F. J., *Latin Hypercube Sampling and the propagation of uncertainty in analyses of complex systems*. Reliability Engineering and System Safety, 81(1):23-69, 2003.

[35] Hörts M., Wohlin C., „A subjective effort estimation experiment." Information and Software Technology, 39(11):755-762, 1997.

[36] Huang Sun-Jen, Lin Chieh-Yi, Chiu Nan-Hsing, "Fuzzy Decision Tree Approach for Embedding Risk Assessment Information into Software Cost Estimation Model." Journal of Information Science and Engineering, 22(2):297-313, March 2006.

[37] Huang X., Capretz L.F., Ren J., Ho D. „A Neuro-Fuzzy Model for Software Cost Estimation." Proceedings of the 3rd International Conference on Quality Software, 2003.

[38] Idri A., Abran A., Khoshgoftaar T., Robert S., "Estimating Software Project Effort by Analogy based on Linguistic Values", Proceedings of the 8th International Symposium on Software Metrics, IEEE computer Society, Ottawa, Canada, June 2002, pp. 21-30.

[39] Idri A., Kjiri L., Abran A. "COCOMO Cost Model Using Fuzzy Logic," Proceeding of the 7th International Conference on Fuzzy Theory &Technology, Atlantic City, New Jersey, February-March 2000.

[40] Jeffery R., Ruhe M., Wieczorek I., "A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data." Information and Software Technology, 42(14):1009-1016, November 2000.

[41] Jensen R.W., Putnam L.H., Roetzheim W., "Software Estimating Models: Three Viewpoints." Cross-Talk, The Journal of Defense Software Engineering, (2):23-29, February 2006.

[42] Jensen R.W., "An Improved Macro-level Software Development Resource Estimation Model," Proceedings of the 5th International Society of Parametric Analysts Conference, St. Louis, Mo. April 1983, pp. 88-92.

[43] Jensen R.W., "A Macro-level Software Development Cost Estimation Methodology." Proceedings of the 14th Asilomar Conference on Circuits, Systems and Computers. Pacific Grove, CA, November 1980.

[44] Johnson P.M., Moore C.A., Dane J.A., Brwer R.S., "Empirically guided software effort guesstimation." IEEE Software, 17(6):51-56, November/December 2000.

[45] Jones T.C., Estimating Software Costs, McGraw-Hill, 1998.

[46] Jørgensen M., "Practical guidelines for better support of expert judgment-based software effort estimation." IEEE Software, 22(3):57-63, May/June 2005.

[47] Jørgensen M., "Realism in Effort Estimation Uncertainty Assessments: It Matters How You Ask," IEEE Transactions on Software Engineering, 30(4):209-217, April 2004.

[48] Jørgensen M., "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models." IEEE Transactions on Software Engineering, 21(8):674-681, 1995.

[49] Jørgensen M., Shepperd M., "A Systematic Review of Software Development Cost Estimation Studies", IEEE Transactions on Software Engineering, 33(1): 33-53, 2007.

[50] Kadoda G., Cartwright M., Shepperd M., "Issues on the Effective Use of CBR Technology for Software Project Prediction." Proceedings of the 4th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development, August 2001, pp. 276-290.

[51] Kalos M. H., Whitlock P. A., *Monte Carlo Methods – Volume I: Basics*. A Wiley-Interscience publication. 1986.

[52] Kitchenham B.A., Pickard L., Linkman S.G., Jones P., "A framework for evaluating a software bidding model." Information and Software Technology, 47(11):747-760 May 2005.

[53] Kitchenham B., Pickard L.M., Linkman S., Jones P.W., ” Modeling software bidding risks.” IEEE Transactions on Software Engineering, 29(6):542- 554, June 2003.

[54] Kitchenham B., “A Procedure for Analyzing Unbalanced Datasets.” IEEE Transactions on Software Engineering, 24(4):278-301, 1998.

[55] Kitchenham B., “Empirical Studies of the Assumptions that Underline Software Cost-Estimation Models.” Information and software Technology, 34(4):211-218, 1992.

[56] L'Ecuyer P., *Efficiency Improvement and Variance Reduction*. Proceedings of The Winter Simulation Conference, 1994.

[57] Lee A., Chun Hung Cheng, Balakrishnan J., “Software development cost estimation: Integrating neural network with cluster analysis.” Information and Management, 34(1):1-9, August 1998.

[58] Li J., Ruhe G. „A comparative study of attribute weighting heuristics for effort estimation by analogy.” Proceedings of the International Symposium on Empirical Software Engineering, Rio de Janeiro, Brazil, 2006, pp. 66-74.

[59] Liang T., Noore A.: “Multistage software estimation.” Proceedings of the 35th Southeastern Symposium on System Theory. March 2003, pp. 232- 236.

[60] Loève M., *Probability Theory*. D. Van Nostrand Company, 1963.

[61] Lother, M., Dumke, R. Point Metrics. Comparison and Analysis. In: Dumke/Abran: *Current Trends in Software Measurement*, Shaker Publ., Aachen, Germany, 2001, pp. 228-267.

[62] MacDonell S.G., “Establishing relationships between specification size and software process effort in CASE environments.” Information and Software Technology, 39(1):35-45, 1997.

[63] MacDonell S.G., Gray A.R., “Alternatives to regression models for estimating software projects.” Proceedings of the IFPUG Fall Conference, Dallas TX, 1996.

[64] Mair C., Shepperd M.J. "An Investigation of Rule Induction Based Prediction Systems", Proceeding of the IEEE ICSE Workshop on Empirical Studies of Software Development and Evolution, May 1999.

[65] McKay M. D., Beckman R. J., Conover W. J., *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code*. Technometrics, 21(2):239-45, 1979.

[66] Mendes E., Watson I., Chris T., Nile M., Steve C., “A comparative study of cost estimation models for web hypermedia applications.” Empirical Software Engineering, 8(2):163-196, 2003.

[67] Menzies T., Port D., Chen Z., Hihn J., Stukes S., “Validation methods for calibrating software effort models”, Proceedings of the 27th International Conference on Software Engineering, May 2005, pp. 587-595.

[68] Miyazaki Y., Terakado M., Ozaki K., Nozaki H., “Robust Regression for Developing Software Estimation Models.” Journal of Systems and Software. 27:3-16, 1994.

[69] Moløkken-Østvold K. J., Jørgensen M., ”Group Processes in Software Effort Estimation.” Empirical Software Engineering, 9(4):315-334, December 2004.

[70] Mukhopadhyay T., Vicinanza S.S., Prietula M.J. “Examining the feasibility of a case-based reasoning model for software effort estimation.” MIS Quarterly, 16(2):155-171, June 1992.

[71] Musilek P., Pedrycz W., Succi G.: “Software cost estimation with fuzzy models.” ACM SIGAPP Applied Computing Review, 8(2):24-29, 2000.

[72] Ohsugi N., Tsunoda M., Monden A., Matsumoto K., "Applying collaborative filtering for effort estimation with process metrics." Proceedings of the 5th International Conference on Product Focused Software Process Improvement, Kyoto, Japan, 3009:274-286, April 2004.

[73] Owen A. B., *Monte Carlo Variance of scrambled equidistribution quadrature.* Journal on Numerical Analysis, 34(5):1884–1910, 1997.

[74] Pawlak Z., Rough Sets: Theoretical Aspects of Reasoning About Data. Kluwer Academic Publishing, 1991.

[75] Pendharkar P.C., Subramanian G.H., Rodger J.A., "A Probabilistic Model for Predicting Software Development Effort," IEEE Transactions on Software Engineering, 31(7):615-624, July 2005.

[76] Putnam L.H., Myers W., Five Core Metrics. Dorset House, New York, 2003.

[77] Putnam L.H., Myers W., "What We Have Learned." Cross-Talk: The Journal of Defense Software Engineering, June 2000.

[78] Reifer D.J., "SoftCost-R: User experiences and lessons learned at the age of one." Journal of Systems and Software, 7(4):279-286, December 1987.

[79] *@RISK 4.5 User's Guide*. Palisade Corporation. October, 2004.

[80] Rodriguez D., Cuadrado-Gallego J.J. and Aguilar J., "Using Genetic Algorithms to Generate Estimation Models." Proceedings of the International Workshop on Software Measurement and Metrik Kongress, Potsdam, Germany, 2006.

[81] Saliby E., *Descriptive Sampling: An Improvement over Latin Hypercube Sampling.* In Proceedings of Winter Simulation Conference, IEEE Press, 1997.Saliby E., *An Empirical Evaluation of Sampling Methods in Risk Analysis Simulation: Quasi-Monte Carlo, Descriptive Sampling, and Latin Hypercube Sampling.* In Proceedings of TheWinter Simulation Conference, IEEE Press, 2002.

[82] Samson B., Ellison D., Dugard P., "Software Cost Estimation Using an Albus Perceptron (CMAC)." Information and Software Technology, 39(1):55-60, 1997.

[83] Sentas P., Angelis L., Stamelos I., Bleris G.L., "Software Productivity and Effort Prediction with Ordinal Regression." Journal of Information & Software Technology, 47(1):17-29, January 2005.

[84] Shan Y., McKay R.I., Lokan C.J., Essam D.L., "Software Project Effort Estimation Using Genetic Programming." Proceedings of the International Conference on Communications, Circuits and Systems, Chengdu, China, July 2002, pp. 1108-1112.

[85] Shepperd M., Cartwright M. "Predicting with sparse data." IEEE Transactions on Software Engineering, 27(11):987-998, 2001.

[86] Shepperd M., Kadoda G., "Comparing Software Prediction Techniques Using Simulation", *IEEE Transactions on Software Engineering*, 27(11):1014-1022, November 2001.

[87] Shepperd M., Schofield C., "Estimating Software Project Effort Using Analogies." IEEE Transactions on Software Engineering, 23(12):736-743, 1997.

[88] Shin M., Goel A.L., "Empirical Data Modeling in Software Engineering Using Radial Basis Functions." IEEE Transactions on Software Engineering, 26(6):567-576, June 2000.

[89] Shukla K.K., "Neuro-genetic prediction of software development effort." Information and Software Technology, 42:701-713, 2000.

[90] Smith R.K., Hale J.E., Parrish A.S., ”An Empirical Study Using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation.” IEEE Transactions on Software Engineering, 27(3):264-271, March 2001.

[91] Sobol I. M., *The Monte Carlo Method*. The University of Chicago Press. Chicago, 1974.

[92] Song Q., Shepperd M., Mair C., “Using grey relational analysis to predict software effort with small data sets.” Proceedings of the 11th IEEE International Software Metrics Symposium. Como, Italy, 2005, pp 35-45.

[93] Spector P. Summated Rating Scale Construction. Sage Publications, 1992.

[94] Srinivasan K., Fisher D., “Machine Learning Approaches to Estimating Software Development Effort.” IEEE Transactions on Software Engineering, 21(2):126-137, 1995.

[95] Stamelos I., Angelis L., “Managing uncertainty in project portfolio cost estimation.” Information and Software Technology, 43(13):759-768, November 2001.

[96] Stamelos I., Angelis L., Sakellaris E., “BRACE: bootstrap based analogy cost estimation.” Proceedings of the 12th European Software Control Metrics, London, UK, April 2001, pp. 17-23.

[97] StatSoft Inc., Statistica 7 Data Miner, http://www.statsoft.com

[98] Stroud, A. H., *Approximate Calculation of Multiple Integrals*. New Jersey: Prentice-Hall, 1971.

[99] Taff L.M., Brochering J.W., Hudgins W.R. “Estimeetings: Development Estimates and a Front-End Process for a Large Project.” IEEE Transactions on Software Engineering, 17(8):839-849, 1991.

[100] Trendowicz A., Heidrich J., Münch J., Ishigai Y., Yokoyama K., Kikuchi N., *Development of a Hybrid Cost Estimation Model in an Iterative Manner*, Proceeding to 28th International Conference on Software Engineering ICSE 2006, Shanghai, China, 2006.

[101] The Standish Group. *CHAOS Chronicles*, West Yarmouth, MA, The Standish Group International, Inc., 2005.

[102] Vose, D., *Quantitative Risk Analysis. A Guide to Monte Carlo Simulation Modelling*, Chichester: John Wiley & Sons, 1996.

[103] Walkerden F., Jeffery R., “An Empirical Study on Analogy-based Software Effort Estimation." Empirical Software Engineering, 4:135-158, 1999.

[104] Walkerden F., Jeffery R., “Software Cost Estimation: A Review of Models, Process, and Practice.” Advances in Computers, 44:59-125, 1997.

[105] Wieczorek I., “Improved software cost estimation - a robust and interpretable modelling method and a comprehensive empirical investigation.” Empirical Software Engineering, 7(2):177-180, 2002.

[106] Wohlin C., Runeson P., Höst M., Ohlsson M., Regnell B., Wesslen A., *Experimentation in Software Engineering. An Introduction*. Boston : Kluwer Academic Publishers, 2000.

[107] Xu Z., Khoshgoftaar T.M. “Identification of fuzzy models of software cost estimation.” Fuzzy Sets and Systems, 145(1):141-163, 2004.

[108] Yang D., Wan Y., Tang Z., Wu S., He M., Li M., “COCOMO-U: An extension of COCOMO II for cost estimation with uncertainty.” In: Wang Q, et al. (eds.) Software Process Change, SPW/ProSim 2006. LNCS 3966, Berlin, Heidelberg: Springer-Verlag, 2006. pp. 132-141.

[109] Zadeh L.A., “Fuzzy Sets.” ’Information and Control”, 8:338-353, April 1965.

[110] Zou T., Mahadevan S., Mourelatos Z., Meernik P., *Reliability analysis of automotive body-door subsystem*. Reliability Engineering & System Safety, 78(3):315-324, December 2002.