Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Software Technology xx (2004) xxx–xxx
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Adaptive fuzzy logic-based framework for software development effort prediction

Moataz A. Ahmed[a], Moshood Omolade Saliu[b,*], Jarallah AlGhamdi[a]

[a]Dept of Info. & Computer Science, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia
[b]Dept of Computer Science, University of Calgary, 2500 University Drive NW, Calgary, AB, Canada T2N 1N4

Received 20 October 2003; revised 19 May 2004; accepted 21 May 2004

Abstract

Algorithmic effort prediction models are limited by their inability to cope with uncertainties and imprecision present in software projects early in the development life cycle. In this paper, we present an adaptive fuzzy logic framework for software effort prediction. The training and adaptation algorithms implemented in the framework tolerates imprecision, explains prediction rationale through rules, incorporates experts knowledge, offers transparency in the prediction system, and could adapt to new environments as new data becomes available. Our validation experiment was carried out on artificial datasets as well as the COCOMO public database. We also present an experimental validation of the training procedure employed in the framework.

Keywords: Soft computing; Effort prediction; Fuzzy logic; COCOMO

1. Introduction

Software cost and schedule estimation supports the planning and tracking of software projects. Effectively controlling the expensive investment of software development is of paramount importance [20,39]. The need for reliable and accurate cost predictions in software engineering is an ongoing challenge [13], because it allows for considerable financial and strategic planning.

Software cost estimation refers to the predictions of the likely amount of effort, time, and staffing levels required to build a software system. A very helpful form of effort prediction is the one made at an early stage during a project, when the costing of the project is proposed for approval. This project costing is derived primarily from requirements specifications documents [18]. However, estimates at the early stages of the development are the most difficult to obtain. The estimates are often the least accurate, because very little detail is known about the project and the product at the beginning.

In this paper, we present an adaptive fuzzy logic (FL) framework for effort prediction. The paper is organized as follows. In Section 2, we discuss the overtime evolution of both algorithmic and non-algorithmic models; Section 3 presents our attributes for evaluating soft computing-based prediction systems, and summary result of our critical study using the attributes. Section 4 presents an adaptive and transparent framework for effort prediction, and the training algorithms implemented. Section 5 discusses the various experiments to realize the framework. Section 6 concludes discussions of our experimental results, and Section 7 points out possible directions for future research.

2. Effort prediction models

Software effort estimation spawned some of the first attempts at rigorous software measurement, so it is the oldest, most mature aspect of software metrics. This section discusses the evolution of both algorithmic and non-algorithmic estimation techniques overtime. We summarize the section by giving the motivation for our work in this research.

* Corresponding author. Tel.: +1-403-210-8412.
E-mail addresses: saliu@cpsc.ucalgary.ca (M. Omolade Saliu), mahmed@ccse.kfupm.edu.sa (M.A. Ahmed), jaralla@ccse.kfupm.edu.sa (J. AlGhamdi).

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.05.004

2.1. Algorithmic models

Boehm was the first researcher to look at software engineering from an economic point of view. He came up with a cost estimation model, COCOMO-81 in 1981, after investigating a large set of data from TRW in the 1970s [10]. Putnam also developed an early model known as SLIM in 1978 [38]. COCOMO and SLIM [11] are both based on linear regression techniques, using data from past projects. Both COCOMO and SLIM take number of lines of code (about which least is known very early in the project) as the major input to their models. Albrecht's function points measures the amount of functionality in a system as described by a specification [11]. A survey on these algorithmic models and other cost estimation approaches is presented by Boehm et al. [2].

Most models rely on accurate estimate of either size of software in terms of line of code (LOC), number of user screen, interfaces, complexity, etc. at a time when uncertainty is mostly present in the project [31].

Algorithmic models such as COCOMO, have failed to present suitable solutions that take into consideration technological advancements [13]. One possible reason why algorithmic models have not proven to provide such solution is because, they are often unable to capture the complex set of relationships (e.g. the effect of each variable in a model to the overall prediction made using the model) that are evident in many software development environments [34]. They can be successful within a particular type of environment, but not flexible enough to adapt to a new environment. Their inability to handle categorical data (that is, data that are specified by a range of values) and most importantly lack of reasoning capabilities (that is, ability to draw conclusions or make judgments based on available data) contributed to the number of studies exploring non-algorithmic methods (e.g. FL).

Section 2.2 discusses some of the non-algorithmic models that are soft computing-based. Soft computing is a consortium of methodologies centering in FL, artificial neural networks (ANN) and evolutionary computation (EC). These methodologies provide flexible information processing capability for handling real life ambiguous situations.

2.2. Non-algorithmic models

Newer computation techniques to cost estimation that are non-algorithmic were sought in the 1990s. Researchers turned attention to a set of approaches that are soft computing-based.

FL with its offerings of a powerful linguistic representation can represent imprecision in inputs and outputs, while providing a more expert knowledge-based approach to model building. A study by Hodgkinson and Garratt claims that estimation by expert judgment was better than all regression-based models [13].

MacDonell et al. [21] explored an expert knowledge-based application of FL to effort prediction. This particular research has evolved into the development of a tool, FULSOME, to assist project managers in making predictions. MacDonell also applied fuzzy modeling to software source code sizing in Ref. [19].

Attempts have been made to fuzzify some of the existing algorithmic models in order to handle uncertainties and imprecision problems in such models. The first realization of the fuzziness of several aspects of one of the best known [17], most successful and widely used model for cost estimation, COCOMO, was that of Fei and Liu [10]. Fei and Liu observed that an accurate estimate of delivered source instruction (KDSI) cannot be made before starting the project. Therefore, it is unreasonable to assign a determinate number for it. Jack Ryder [31] investigated the application of fuzzy modeling techniques to two of the most widely used models for effort prediction; COCOMO and the Function-Points models, respectively. Idri and Abran [14] applied FL to the cost drivers of intermediate COCOMO model. The application of FL to represent the *mode* and *size* as input to COCOMO model was later presented by Musilek et al. in Ref. [24]. Musilek et al. presented a two-stage implementation called simple F-COCOMO model and augmented F-COCOMO model, respectively.

FL has also offered itself as a useful tool to aid other techniques for software cost estimation like analogy. Similarity between projects is often used when estimating software effort by analogy. Various authors have put forward various proposals for means of deriving similarity as input to the estimation process; the *nearest neighbor algorithm* [34] is one such approach. This algorithm cannot handle projects attributes described by categorical variables. Idri and Abran [15,16] proposed an alternative approach using FL to deal with this limitation.

Venkatachalam [40] investigated the application of ANN to cost estimation. Other latest works using neural networks (NN) in cost estimation are reported in Refs. [3,4]. ANN is able to generalize from trained data set. Over a known set of training data, an ANN learning algorithm constructs mappings that fit the data, and fits previously unseen data in a reasonable manner as well [38].

A marriage between NN and FL, Neurofuzzy, was introduced into cost estimation in Ref. [13]. A Neurofuzzy system can combine the linguistic attributes of a fuzzy system with the learning and modeling attributes of a NN.

EC has also recently found its usefulness in software effort estimation. Burgess and Lefley [6] applied genetic programming (GP) to software effort estimation. EC simulates evolution on a computer.

Chulani et al. [7] applied Bayesian analysis in calibrating the 1998 version of COCOMO II model to 161 data-points. When compared with the 1997 calibration done using multiple regressions, the Bayesian approach was adjudged to perform better and more robust. Bayesian analysis was also used in the calibration of the 2000 version of

COCOMO II by Boehm et al. [2]. The result of this was a higher predictive accuracy.

In summary, FL has been proposed to model imprecision present in the variables that make up the COCOMO model; and the imprecision present in software effort prediction in general. However, there is still much uncertainty as to what prediction technique suits which type of prediction problem [35]. Choosing between the various techniques is an arduous decision that requires the support of a well-defined evaluation scheme to rank each prediction technique as it applies to any prediction problem. A sample evaluation scheme is given in Section 3 of this paper.

Section 3 reveals that a problem with existing techniques is the absence of transparent framework that incorporates expert knowledge, information from historical project data, imprecision handling and adaptability. The existence of such framework would explore most information source available to make predictions that could gain the confidence of project managers.

3. Classification attributes for soft computing-based techniques

A software project manager will need to trust predictions made by prediction techniques, if these techniques are to be deployed in practice [18]. Prediction can be viewed from three different perspectives: the prediction problem (e.g. estimating the cost of development/testing, the duration of development/testing, etc.), the particular problem at hand (a specific case with its own characteristics, e.g. a supermarket software), and finally the prediction technique (e.g. algorithmic model, analogy, FL, etc.). Assessing a prediction technique, in our own view, should not only be based on accuracy of estimates made, but also on other attributes of the underlying prediction model. To the best of our knowledge, assessing prediction systems in the literature have been based on the characteristics of the dataset (i.e. the data on which the prediction is carried out) and the accuracy of the model only [35].

There is still much uncertainty as to what prediction technique suits which type of prediction problem [35]. This type of uncertainties is what we desire to address by proposing set of attributes to aid the assessment and choice of prediction system. Our critical survey result that concludes this section was guided by the set of attributes discussed in the sequel. We propose this set of attributes, which was developed based on surveying the literature, for assessing, classifying, and comparing soft computing-based effort prediction systems:

Underlying model. The underlying model specifies whether the soft computing approach to effort prediction is based on an existing algorithmic cost estimation model like COCOMO, SLIM, etc. or based on other models like expert judgment, analogy, etc. Empirical research has indicated that expert judgment coupled with prediction systems sometimes outperform either prediction systems or expert judgment alone [23]. Based on this observation, it might not make much sense to keep everything to a single model. For example, if the underlying model were known not to always perform accurately, this would affect any variant of it that is implemented.

Trainability. Trainability is the ability of a prediction system to learn the relationships between features and adapt during training. This attribute is what has generally been referred to as adaptability in many of the approaches surveyed.

Adaptability. This attribute describes the ability and ease of the prediction system to adjust to new environments as new information and knowledge are supplied. Trainability does not translate to adaptability, as a system could be trainable but not adaptive. Adaptability subsumes trainability.

Sensitivity. This attribute refers to the responsiveness to changes in input data, and the type of input data it can handle (e.g. numeric data, categorical data). Responsiveness to changes in input data assesses the effect an imprecision in input to the model has on the effort estimate produced. For example, we desire to know how well the system can accommodate an error involving size supplied as 900 KLOC as opposed to the actual 850 KLOC.

Aspect coverage. This refers to the ability of the approach to cover wide range of *aspects* of the development process and environment. For example, whether the effort prediction system takes into consideration the following; *reuse*, *capability-maturity model level*, etc.

Spectrum coverage. This attribute refers to the coverage of different types (classes) of systems, e.g. organic, semi-detached and embedded systems, as proposed by Boehm in Ref. [1]. If a prediction system is not sophisticated, it might not be able to cover the whole spectrum. A prediction system might still be able to model a software project inherently made up of different classes without necessarily breaking the project into different groups, although such systems may be too complicated.

Implementation technique. We define this attribute to capture the implementation approach taken. An implementation approach using soft computing can be as simple as a straightforward application of an underlying model by applying a single soft computing approach, e.g. fuzzifying input/output, or a more sophisticated implementation technique that explores/combines various capabilities of the soft computing methods used (e.g. FL, NN, neurofuzzy (NF), neuro-genetic (NG), etc.)

Input data. This attribute identifies the type of input data required by the effort prediction system to perform estimates, e.g. lines-of-code. It also reflects the ease of getting the input data and the accuracy in making reasonable estimates. Input data is simply the input required to make estimates using the prediction system, but not to develop the prediction system.

Knowledge acquisition and data source. This refers to the mode of knowledge acquisition considered in developing (i.e. training/adapting) the prediction system, the source of data required and how reliant the system is on the data source. The mode of knowledge acquisition could either be *manual*, with users being the source of the knowledge or *automatic* (based on perceived relationship between the data through learning). For example, a system that relies on users to supply rules in a FL-based approach is said to exhibit manual knowledge acquisition. The data source can be either from historical or simulated data.

Complexity of the model. This attribute refers to the amount of effort or size (e.g. number of neurons, number or rules, etc.) required for building and/or using the prediction system. This attribute reflects the efficiency of the prediction system. A model that is not practical or rather difficult to use might not be a good model. For example, NN are known to give good approximations, but they might be overly complex and require considerable effort and expertise [23].

Accuracy. Accuracy is the attribute of a prediction system that reflects its effectiveness. A software manager who wants to use a prediction model would desire to use an accurate one.

Transparency. Transparency of a prediction system reflects the visibility of the prediction process to the software engineer/expert. Interaction or collaboration between the prediction system and the end-user/expert is of great importance, especially for maintenance purposes. If a system is transparent, an expert can easily evaluate and add his own knowledge to improve accuracy of the model, because it would be possible to see and understand the processes involved. Empirical research has indicated that experts coupled with prediction systems outperform either prediction systems or experts alone [23].

Extendibility. Extendibility reflects the ability of a prediction system to accommodate changes to its model, in that it will be useful for predicting effort required for other activities of software development, e.g. maintenance, testing, etc. A prediction system that uses an underlying model in such a way that the prediction process expects a specific type of input, might not be useful on extending it to other activities for which such inputs are not defined.

The assessment attributes presented offer more beyond their usefulness in carrying out comparison of prediction systems. They can serve as guidance to researchers attempting to develop prediction systems for effort estimation using soft computing.

This set of attributes can serve as a fundamental groundwork of what needs to be measured when assessing and comparing prediction systems. A set of metrics can be built on top of this groundwork to quantitatively assess prediction systems. This type of metric would help a software engineer in decision making as to what prediction system to use and when to use it. An approach to developing such metrics was presented with a sample rating scheme in our previous work in Ref. [32]. Meanwhile, we admit that some of these attributes may be difficult to measure in a systematic manner.

Some other researchers have proposed attributes to assess effort prediction models, but not much attention was given to soft computing. Typical works in this area include that of Boehm [1], Briand and Wieczorek [5], Burgess and Lefley [6], Gray and MacDonell [12], and Mair et al. [23]. Most of the attributes proposed by these researchers are subsumed in our attributes.

In concluding this section, Table 1 gives a summary of our critical study of important soft computing-based

Table 1
Evaluation of prediction techniques

Work	Model	Approach	Limitations
Musilek et al. [24]	FL	Fuzzification of COCOMO nominal effort model	Not adaptive No fuzzy rules Mode modelled as singletons
Idri and Abran [14]	FL	Fuzzification of COCOMO Cost Drivers	Not adaptive Key project feature ignored (SIZE)
MacDonell et al. [20–22]	FL	FULSOME Tool	Not adaptive (rulebase need to be reconstructed)
Wittig and Finnie [42]	NN	Training using historical/simulated data	Trainable, not adaptive Imprecision not handled No Transparency
Boetticher [3,4]	NN	Training using historical data	Trainable, not adaptive Imprecision not handled No Transparency
Burgess and Lefley [6]	GP	Effort estimation formulated as symbolic regression problem	Not adaptive Imprecision not handled No total transparency
Hodgkinson and Garratt [13]	NF	Used NN learning to extract fuzzy rules from data	Trainable, not adaptive No total transparency
Shukla [37]	NG	Genetically trained NN using historical data	Trainable, not adaptive Imprecision not handled No transparency

prediction systems using the attributes we have presented in Section 2.

The considered works use models that include, FL, NN, GP, and hybrids like NF and NG. A more detailed evaluation is contained in Ref. [32].

Our critical evaluation reveals that no single soft computing-based software effort prediction system exists that incorporates the following:

1. Tolerance of imprecision
2. Incorporating expert's knowledge in a well-defined manner
3. Total transparency in the prediction system
4. Ability to explain prediction results through rules or other means
5. Adaptability to cope with continuous change in development technologies and environments.

Properly addressing these problems would position soft computing-based prediction techniques as models of choice for effort prediction, considering the promising features already present in them.

4. Adaptive and transparent framework

Most significant of the problems identified in concluding our critical study is the lack of adaptive soft computing-based effort prediction systems that provide complete transparency to the prediction system building strategies. Thus, experts would not be able to easily augment with their knowledge while building and using the prediction system.

In addition, efforts at handling the imprecision problem present in one of the most widely used algorithmic model, COCOMO, has not been appropriate, since the model was not addressed as a whole. That is, imprecision in input is separately modeled for the nominal effort part of the model and cost drivers. Thus, integrating the individual component of the model into a single prediction system remains an open question.

In order to address these problems we propose a FL-based framework that is built upon an existing cost estimation model—COCOMO. A major justification for using COCOMO is that, while many traditional models have been said to perform poorly when it comes to cost estimation, COCOMO'81 is said to be most plausible [24], best known [17], and most cited [13] of all traditional models. For detailed justification of the assertions above, the reader is advised to consult the works by Musilek et al. [24], Kirsopp and Shepperd [17] and Hodgkinson and Garratt [13]. Devnani-Chulani reported that over a dozen commercial COCOMO'81 implementations are available [9]. The original COCOMO database is also readily available for validation.

Our framework to be presented later in this section addresses these noted problems by:

- Providing a well-defined and transparent approach to accommodate expert knowledge
- Proposing a well-defined procedure to fuzzify and integrate the various components of the COCOMO model that captures imprecision, and
- Implementing training algorithms to incorporate adaptability

4.1. The intermediate COCOMO model

The COCOMO model is a set of three models: basic, intermediate, and detailed. The models depend on the stage of software development and the level of information available. The basic version is used for quick, early, and rough estimates of effort. The intermediate and detailed versions include more information in the form of cost drivers. The detailed COCOMO model assumes that the influence of the cost drivers is phase-dependent, thus requiring the availability of much detailed information than the intermediate version.

Idri and Abran [14] reported that intermediate COCOMO model is the most widely used version. According to the authors, intermediate COCOMO model has estimation accuracy that is greater than the basic version, and at the same time comparable to the detailed version. COCOMO model takes the following as input: (1) the estimated size of the software product in thousands of Delivered Source Instructions (KDSI) adjusted for code reuse [1,8]; (2) the project development mode given as a constant value B (also called the scaling factor); and (3) 15 cost drivers. The development mode depends on one of the three categories of software development modes: organic, semi-detached, and embedded. It takes only three values, {1.05, 1.12, 1.20}, which reflect the difficulty of the development. The estimate is adjusted by factors called *cost drivers* that influence the effort to produce the software product. Cost drivers have up to six levels of rating: *Very Low*, *Low, Nominal, High, Very High*, and *Extra High*. Each rating has a corresponding real number (effort multiplier), based upon the factor and the degree to which the factor can influence productivity.

The estimated effort in person-months (PM) for the intermediate COCOMO is given as:

$$\text{Effort} = A \times [\text{Size}]^B \times \prod_{i=1}^{15} \text{EM}_i \quad (1)$$

The constant A in Eq. (1) is also known as productivity coefficient. The scale factors for the different modes are given in Table 2. These scale factors are based solely on the original set of project data used in developing the COCOMO model.

The effort multipliers corresponding to the cost drivers are incorporated into the effort estimation formula by multiplying them together. The numerical value of the ith cost driver is EM_i and the product of all the multipliers is called the estimated adjustment factor (EAF). The actual effort in PM, PM_{total} is the product of the nominal effort

Table 2
COCOMO mode coefficients and scale factors values

Mode	A	B
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.2

(i.e. effort without the cost drivers) and the EAF, as given below:

$$\mathrm{PM}_{\mathrm{total}} = \mathrm{PM}_{\mathrm{nominal}} \times \prod_{i=1}^{15} \mathrm{EM}_i \quad (2)$$

4.2. The proposed framework

Vagueness is present in all parameters of the COCOMO model. The exact size of software project to be developed is difficult to estimate to precision at an early stage of the development process. There is no consideration given to software projects that do not exactly fall into one of the three identified modes. In addition, the cost drivers are categorical (e.g. high, low, etc.), but determining the category that applies to a project for most cost driver depends on crisp values estimates that may not entirely fall under a single categorical class. In all these input parameters, fuzzy sets can be employed to handle the imprecision.

Various attempts have been made to address the fuzzy nature of some aspects of the COCOMO model, with prominent ones including Idri and Abran [14], Musilek et al. [24], and Pedrycz et al. [29].

However, these works have not looked into the integration of the nominal effort and the cost drivers of the COCOMO model. Our adaptive framework uses intermediate COCOMO as the base cost model to address the major problem of data scarcity and fuzzy rules training. Data scarcity refers to the unavailability of industrial data. We have been able to generate artificial datasets using the COCOMO model. A detail description of artificial data generation is given in Section 5. The framework is shown in Fig. 1.

The framework integrates the two components of the COCOMO model as given in Eq. (2): the nominal effort as calculated using the basic part of the model, and the effort adjustment using the effort multipliers. The framework allows fuzzy and expert knowledge incorporation.

We developed and trained fuzzy rules using the model equation (and expert knowledge) for the basic component at the top to estimate nominal effort, and fuzzified the cost drivers represented at the bottom part of Fig. 1. Independent training of the two components is valid because the effect of one multiplier is independent of the effect of others. Pfleeger observed the need for this independence of cost drivers in Ref. [30]; we have equally carried out an analytical justification in Ref. [33].

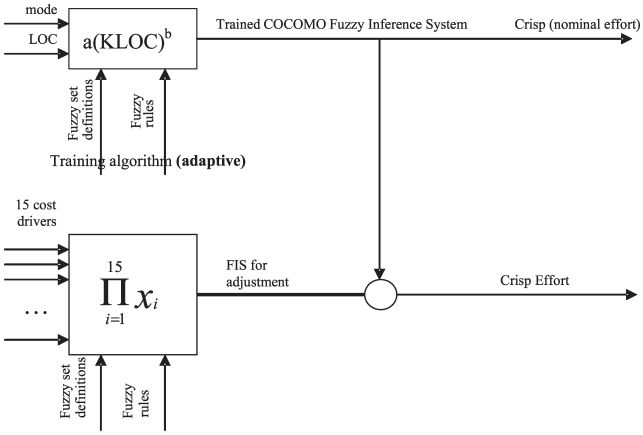

Fig. 1. Adaptive fuzzy logic framework using COCOMO.

Thus, we can fuzzify and train the nominal part of the model equation independent of the cost drivers. Effects of the drivers on efforts are independent and can be aggregated with the predicted nominal effort. The fuzzy sets definitions take care of imprecision in input to the effort prediction model. Informal description of our approach for fuzzifying the two components that constitute the overall effort is given below.

4.2.1. Part I: fuzzy COCOMO for nominal effort

Considering the basic part of the model equation (Eq. (2)) for calculating nominal effort [1], we get

$$E_{\mathrm{nominal}} = (A \times [\mathrm{Size}]^B)$$

Handling the imprecision in input supplied for size requires that size of software project be redefined as a fuzzy number, instead of crisp number, using suitable fuzzy sets. Thus, the size of a software project can be specified as a range of possible values and the closeness of each value to the actual value depends on the degree to which the value is a member of the fuzzy set describing the fuzzy number. Similarly, given the values for A and B as in Ref. [1], we define fuzzy sets for each mode of development such that each adjacent fuzzy set overlaps with its neighbors. The overlap will exploit the power of FL to enable us handle development projects that fall between two modes (a situation not possible in the COCOMO model). Sample artificial datasets are generated for size and the corresponding *effort* is calculated using the COCOMO model equation. For each selected size and cost calculated, membership functions are defined. We then proceed to formulate rules based on the relationship between size and effort; and train the fuzzy inference system (FIS) developed using the artificial dataset.

Implementing a fuzzy system requires that the different categories of the different inputs be represented by fuzzy sets, which in turn is represented by membership functions. For our problem, a natural membership function type that readily comes to mind is the triangular membership functions. It is a three-point function,

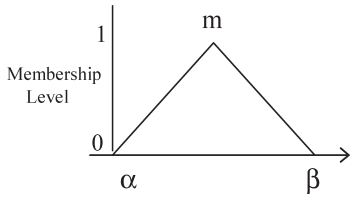

Fig. 2. A triangular membership function.

defined by minimum (α), maximum (β) and modal (m) values, that is, TMF(α, m, β), where ($\alpha \leq m \leq \beta$), Please refer to Fig. 2 for a sample triangular membership function, showing α, m, and β with center m, and support $[\alpha, \beta]$.

The fuzzy sets definitions for the mode of development appear in Fig. 3 below. For example, the middle triangular fuzzy set can be used to represent the fuzzy number (*approximately* semi-detached), with center 1.12, and support [1.05,1.20].

In addition, for size of software fuzzification, we have fuzzy sets as shown in Fig. 4 (triangular membership function) in the simplest case. The hypothetical effort fuzzy sets will correspondingly be as given in Figs. 5 (the unit of effort is in man-months).

Our rules formulated, based on the fuzzy sets of modes, sizes and efforts appear in the following form:

IF mode is organic AND size is $s1$ THEN cost is $c11$
IF mode is semi-detached AND size is $s1$ THEN effort is $c21$
IF mode is embedded AND size is $s1$ THEN effort is $c31$
IF mode is organic AND size is $s2$ THEN effort is $c12$
IF mode is semi-detached AND size is $s2$ THEN effort is $c22$
IF mode is embedded AND size is $s2$ THEN effort is $c32$
…
IF mode is m_j AND size is s_i THEN effort is c_{ji} $(1 \leq i \leq n, 1 \leq j \leq 3)$

Where m_j are the fuzzy values for the fuzzy variable mode, s_i $(1 \leq i \leq n)$ are the fuzzy values for the fuzzy variable size, and C_{ji} $(1 \leq i \leq n, 1 \leq j \leq 3)$ are the fuzzy values for fuzzy variable cost (effort).

The fuzzy membership functions of these rules are trained to improve their prediction quality (i.e. a measure of the prediction accuracy). In this paper, the prediction quality, PRED(t), is calculated as the percentage of the number of predicted values that fall within a threshold t of their actual values. The resulting trained FIS can be used to predict nominal effort early in the software life cycle, when a detailed knowledge of the cost drivers cannot be ascertained.

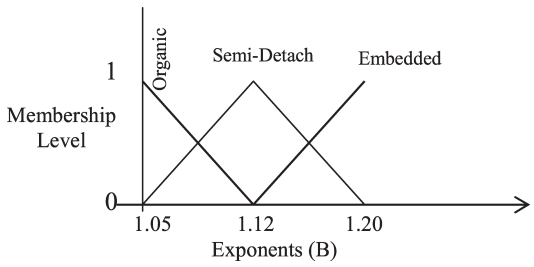

Fig. 3. COCOMO scale factors fuzzification.

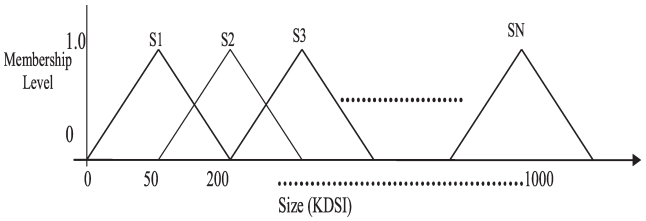

Fig. 4. Fuzzy sets for hypothetical software sizes.

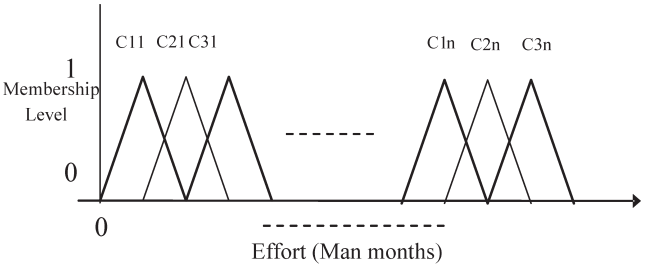

Fig. 5. Fuzzy sets for hypothetical software efforts.

4.2.2. Part II: fuzzy adjustment factors

In the case of cost drivers, we consider each cost driver differently, since their definitions are based on different criteria. Each cost driver originally defined in Ref. [1], are based on rating scales of 'low', 'high', etc. and for each rating, we have corresponding multiplier that specifies the effect of the cost driver on nominal effort estimated.

For incorporating the cost drivers into our framework, we define fuzzy sets for each linguistic values, 'Low', 'Very Low', etc. as it applies to each cost driver. Then rules are formulated using the cost drivers in the antecedent (i.e. the IF part of the rule) and their effects on effort in the consequent (i.e. the THEN part of the rule.). In our framework, each cost driver has its own FIS. The defuzzified values from each of the FIS are aggregated as effort adjustment factor (EAF) to adjust the predicted output from the trained FIS for nominal effort.

For instance, let's consider the TIME (computer turn-around time) cost driver. In this case, we represent the ratings shown in Table 3 using the fuzzy sets shown in Fig. 6.

Table 3
The TIME cost driver range definition

Nominal	High	Very high	Extra high
≤50% use	70%	85%	95%

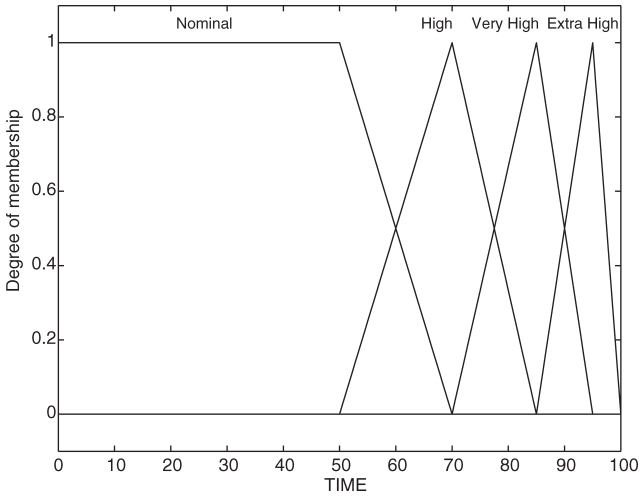

Fig. 6. Antecedent MFs for the FIS of TIME Cost driver.

Similarly, the effort multipliers in Table 4 are represented using fuzzy sets shown in Fig. 7.

From Figs. 6 and 7 we can have rules of the form:

If TIME is Nominal then EFFORT is Unchanged
If TIME is High then EFFORT is Increased
If TIME is Very High then EFFORT is Increased Significantly
If TIME is Extra High then EFFORT is Increased Dramatically

In concluding our presentation of the framework, it is worth noting that our rules formulated for the nominal effort part is required to be trained using an adaptive training algorithm. The algorithm will be able to evolve the relationship between features/factors involved. The adaptive training approach and the algorithms are presented in the subsections that follow. The rules formulated for the cost drivers part do not have to be trained. These rules are simply developed into FISs. However, the membership functions definition and rules formulation are open to experts' knowledge, because our approach is transparent.

Table 4
The TIME cost driver range definition

Nominal	High	Very high	Extra high
1.00	1.11	1.30	1.66

4.3. Adaptive training approach

The training strategy requires building learning capabilities into our FL framework so that the system can learn the importance of the input features and their relationships with effort.

Let us consider once again the nominal effort component of the framework for effort model. Our rules formulation informally discussed in Section 4.2 follow the Mamdani

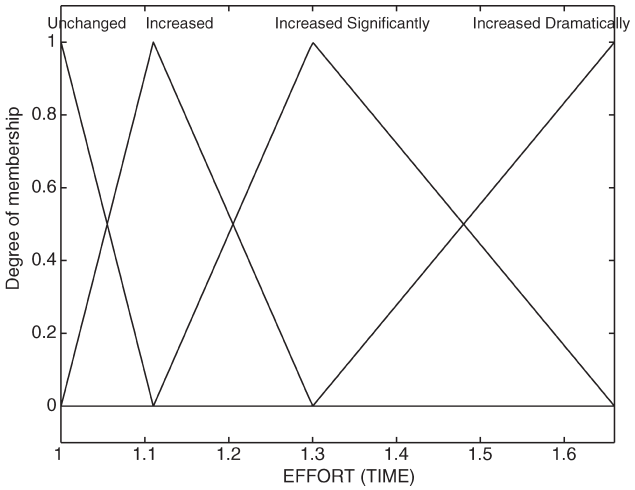

Fig. 7. Consequent MFs for the FIS of TIME cost driver.

max–min fuzzy reasoning. This reasoning can be transformed to fuzzy perceptron structure introduced by Nauck et al. [25,26] in a straightforward manner.

We have rules of the form:

IF MODE is M_j AND SIZE is S_i THEN EFFORT is C_{ji}

Using the Mamdani max–min inference system to evaluate the complete set of rules, the effort derived from the rules is given as:

$$C_{ji}(\text{effort}) = \max_j \{\min\{M_j(x), S_i(y)\}\} \tag{3}$$

where

$$M_j(x) : \Re \rightarrow [0,1], \text{ and also } S_i(y) : \Re \rightarrow [0,1]$$

Eq. (3) is the fuzzy reasoning for evaluating a rulebase to get the predicted output. We have used the fuzzy perceptron learning structure (similar to NN perceptron) [25] to represent the fuzzy reasoning. This representation helps to preserve the meaning of the relationships between the variables, during and after training.

Suppose we derive the following three rules:

R1: IF MODE is M_1 AND SIZE is S_1 THEN EFFORT is C_{11}
R2: IF MODE is M_2 AND SIZE is S_1 THEN EFFORT is C_{21}
R3: IF MODE is M_3 AND SIZE is S_1 THEN EFFORT is C_{31}

The fuzzy perceptron representation of the rules using the Mamdani fuzzy reasoning system is given in Fig. 8.

$\mu_i^{(1)}$, $\mu_i^{(2)}$ are membership functions of the mode and size input variables, respectively, while v_{ij}s are fuzzy sets of the effort output variable. M_j, S_i and C_{ji} are linguistic terms (Fig. 9). Considering rule R1, for instance, M_1, S_1 and C_{11} are linguistic terms represented by the fuzzy sets $\mu_1^{(1)}$, $\mu_1^{(2)}$ and v_{11}, respectively.

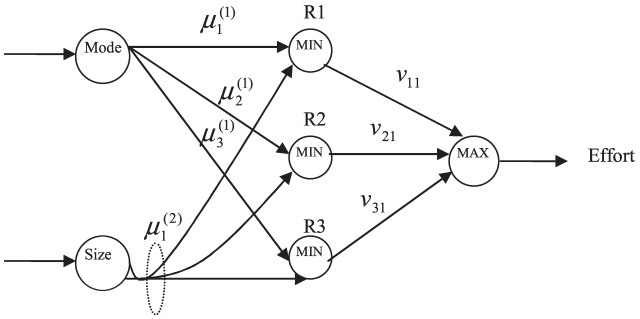

Fig. 8. Cost model rules represented using fuzzy perceptron structure.

From the representation above, the architecture of the fuzzy perceptron is determined by the rules and the fuzzy sets of the underlying problem. This representation retains the transparency in the rules formulated, and retains the built-in fuzzy reasoning. The rules in the rulebase are transparently formulated or updated, and not evolved from data after training as usually done for NF systems that use NN learning algorithms directly. This transparency allows experts to augment with their knowledge by adding new rules. Rules may also be deleted if they are adjudged to consistently perform poorly. It is worth noting that, inconsistencies cannot arise in our rulebase.

The adaptive training algorithms for the framework will encourage modularity of knowledge in adapting to different environments. Adaptability in this context refers to the changing parameters of the MFs in order to accommodate new data. A detailed explanation is given in Section 4.4. Modularity is also achieved in the sense that the whole rulebase and knowledge are not reconstructed or destroyed each time the framework is deployed to different environments.

4.4. The training algorithms for the framework

We built our training algorithm on the generic fuzzy perceptron architecture's algorithm by Nauck et al. [25,26], adapting the algorithm as appropriate for our problem.

The training implementation involves these steps:

1. *Structure-Learning Phase* (for building the IF–THEN rules using the knowledge built into the COCOMO model)
2. *Parameter-Learning Phase* (for tuning the membership functions and rules to optimize)

The structure-learning phase involves building a full rulebase to be optimized during training of the membership functions. Our proposed approach for building a rulebase of the FIS for nominal effort prediction is a special case of the five-step procedure proposed by Wang and Mendel [41] for generating fuzzy rules from numerical data pairs.

An outline summary of the five-step procedure by Wang and Mendel is given as follows:

(i) Divide the input and output spaces into fuzzy regions
(ii) Generate fuzzy rules from given data pairs
(iii) Assign a degree to each rule
(iv) Create a combined fuzzy rule base
(v) Determine a mapping based on the combined fuzzy rule base

Since our rules are not just generated from given data pairs, but guided by the prior knowledge embedded in the COCOMO model, we will only adopt and modify Step (i) of Wang and Mendel for partitioning the size input variable into fuzzy regions using fuzzy sets. The prior knowledge available makes the remaining steps not applicable to our approach. We only need to know the parameters of the MFs of the regions in order to formulate rules. In Wang and Mendel approach, the degree to which data pairs supplied belong to each MF determines the rules to be formulated, because no prior knowledge is available. The steps to build the initial FIS are described by Algorithm 1.

Algorithm 1 is guided by the prior knowledge already embedded in the cost model, which avoids the computational overhead of searching for spurious rules after the rulebase is created. Our rules are reflection of the actual relationships between the variables.

Algorithm 1
Building initial FIS

Step 1: Defining the input variables membership functions
1. Define fuzzy sets for mode input variable (intuitive from mode classification) using triangular membership function, or any other shape of membership functions that is applicable, like Gaussian MFs
2. For the size input variable, suppose the domain interval is $[s-, s+]$, e.g. (1–100 KDSI), where the domain interval means that most probably the size variable will assume values that lie in this interval. Divide the domain interval into $2N + 1$ region, and assign each region a fuzzy membership function. Fig. 9 shows an example where the domain interval is divided into five regions ($N = 2$). The shape of each membership function is triangular in this case—one vertex lies at the center of the region and has membership degree value of 1; the other two vertices lie at the centers of the two neighboring regions, respectively, and have membership degree values equal to zero. That is, each MF is defined as TMF(α, m, β), with center m and support $[\alpha, \beta]$. The membership degree in TMF of center m is 1, and those of α and β are 0

Step 2: Defining the output variables membership functions
1. For the output variable, effort, the parameters of each MF of a selected size MF in Step 1 is plugged into COCOMO nominal effort model to calculate the parameters of the corresponding effort, for each of the three modes. This implies that, for every size MF, we will have three different membership functions corresponding to three regions
2. Repeat (1) as many as the number of input MF we have for size

Step 3: Formulating the rules and populating the rulebase
1. Using the prior knowledge embedded in COCOMO model, we formulate a rule reflecting the relationship between corresponding mode, size and effort MFs selected. This is repeated for as many as the number of effort MFs created in Step 2. Thus, we will have $3(2N + 1)$ regions and MFs in the output. Similarly, we will have $3(2N + 1)$ fuzzy rules in the rulebase of the fuzzy system

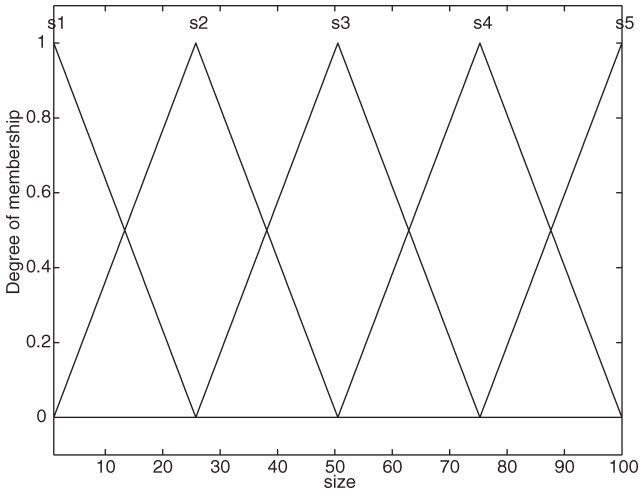

Fig. 9. Divisions of the input variable SIZE into fuzzy regions and the corresponding membership functions.

Algorithm 2
Generating artificial datasets

1. Generate random numbers for a desired number, say K, of unique sizes in the domain interval $[s-, s+]$ considered in Algorithm 1
2. For every number generated in (1), randomly select one of the three development modes and calculate corresponding effort value using the nominal effort model. Each data pattern of the K data points consists of values for size and mode as input, and effort as target
3. Partition the K data-points into training and validation datasets. The training datasets consist of two-third of the entire dataset while the remaining one-third is left for testing after training

The objective of the parameter-learning phase is to adjust parameters of the FIS such that, the error function during training reaches minimum or is less than a given threshold [27,36]. The error measure is not only used to guide the learning process, but also to evaluate the performance of the final model. Training is achieved by adapting the parameters of the membership functions and rules in the input/output layers.

Algorithm 2 describes our approach to generating artificial dataset for training and validation.

The algorithms for fuzzy set learning in a Mamdani-type fuzzy system, originally presented in Refs. [28,29] follow this four-step procedure:

1. Choose a training sample and propagate the input vector across the network to get the output.
2. Determine the error in output, and the error gradient in all the other layers.
3. Determine the parameter changes for the fuzzy weights and update the fuzzy weights.
4. Repeat until the fuzzy error is sufficiently small after an epoch is complete.

We have modified this algorithm where applicable to suit our problem. The modification of the MFs of rules is based on the extent of contribution of each rule to the output.

The fuzzy set learning algorithm (Algorithm 3) uses the following notations:

1. L : a set of training data with $|L| = s$, where patterns $p \in R_n$ as input is mapped to a target $t \in R$.
2. $(p, t) \in L$: a training pattern consists of an input vector $p \in R_n$ and target $t \in R$.
3. $A_r = (\mu_r^{(1)}, \ldots, \mu_r^{(n)})$ the antecedent of rule R_r.
4. 4. $A_r(p)$ denotes the degree of fulfillment of rule R_r (with antecedent A_r) for pattern p, i.e. $A_r(p) = \min\{\mu_r^{(1)}(p_1), \ldots, \mu_r^{(n)}(p_n)\}$.
5. $\mu_r^{(i)}$: a fuzzy set of input variable x_i $(i \in \{1, \ldots, n\})$, $(x_1 = \text{mode}, x_2 = \text{size})$ that appears in the antecedent of fuzzy rule $R_i (r \in \{1, \ldots, k\})$.
6. v_r : a fuzzy set of output variable y (effort) that appears in the consequent of fuzzy rule R_r.
7. δ : is the learning rate of the training algorithm.

Algorithm 3 implements the main loop of the training procedure. In each loop, the algorithm propagates a training pattern, determines the output of the fuzzy system. Using the output error, the algorithm computes parameter updates of the antecedent and consequent MFs, as outlined in the four-step procedure above.

The main information derived from the output error value is whether the contribution of a fuzzy rule to the overall output values should be increased or decreased. The actual modification of the consequent and antecedent MFs is based on some heuristics in the other two algorithms (i.e. Lines 7 and 13 in Algorithm 3). These heuristics are same as in the original algorithms, which is discussed by Nauck in Ref. [27].

The learning algorithm is presented below.

Algorithm 3
Fuzzy set training in Mamdani-type fuzzy system

1. repeat
2. for all patterns (p, t) ∈ L do
3. propagate the next training pattern (p, t);
4. E = (tj - oj)/outputRange (*normalize error*)
5. for each rule Rr with Ar(p) > 0 do
(* Consider only the rules that fire*)
6. for all v_r do (*modify all output fuzzy sets *)
7. ComputeConsequentUpdates(v_r , E, A_r(p), t);
8. Update (v_r);
9. end for
10. $E_r = (2.v_r(t) - 1).|E|$;
11. (*modify antecedent MF with min. membership degree of all MFs*)
12. $j = \min_{i \in \{1, \cdots, n\}} \{\mu_r^{(i)}(p_i)\}$;
13. ComputeAntecedentUpdates $(\mu_r^{(i)}, E_r, p_i)$;
14. Update $(\mu_r^{(j)})$;
15. end for (* end for each rule *)
16. end for (* end for all training patterns *)
17. Calculate RMSRE and test for end criterion
18. until (* end for each epoch *)

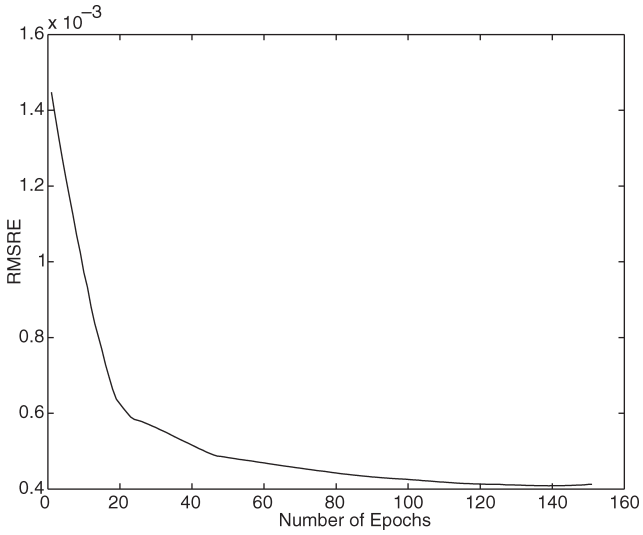

Fig. 10. The error graph during training.

In Algorithm 3 above, we normalized the error E in output, dividing it by the range of the domain interval of the output variable as given in *Line 4*. The idea behind our normalization is to tolerate small errors, and make the value of the error relative to the value of the output variable. This is necessary because training the fuzzy sets takes the ranges of the individual variables into account. If the output variable is using very large values, the size of delta computed is equally so large. In order to be completely independent of the ranges of all the variables, we use the normalization of the error. The training algorithm would therefore successfully operate on any range of input/output variables in the dataset, and there would be no need to normalize the dataset between [0,1] before training, as normally done in NN. This means that the error is bounded and not overemphasized for large errors in output.

The modification factor of the consequent MFs is computed by using the *error*, E, in predicted output for *effort*. The modification factor of the antecedent MFs, on the other hand, is computed from a factor of E (see Line 10 in Algorithm 3).

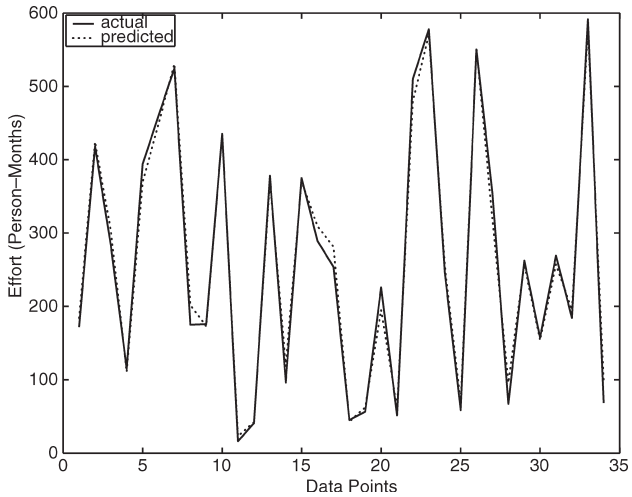

Fig. 11. Prediction of effort using the testing dataset for validation.

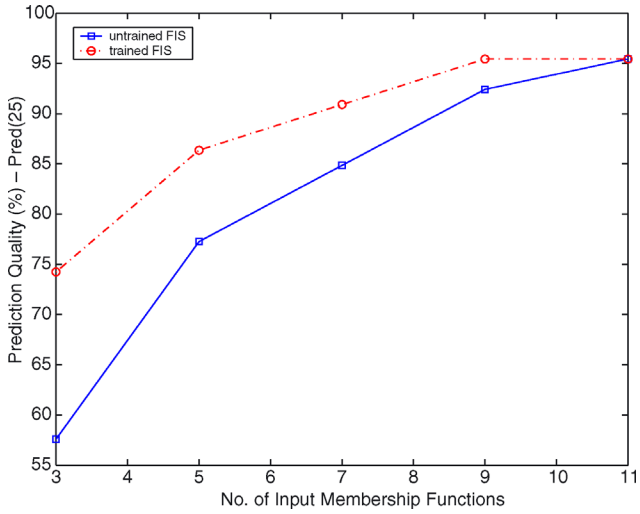

Fig. 12. The prediction quality of untrained and trained FIS using different number of MFs.

The heuristic for modifying the output of a fuzzy system takes the defuzzification procedure into consideration The defuzzification procedure used is the centre of area (COA) [28].

5. Experiments

We have conducted some experiments for the purpose of developing and training FIS for effort (cost) prediction using our proposed approach. The datasets used for the experiments are artificial datasets randomly generated following the procedure discussed in Section 4.

Our experiments basically involve implementation of the training algorithm. During the experiments, we explored different strategies to train the fuzzy system as reported in

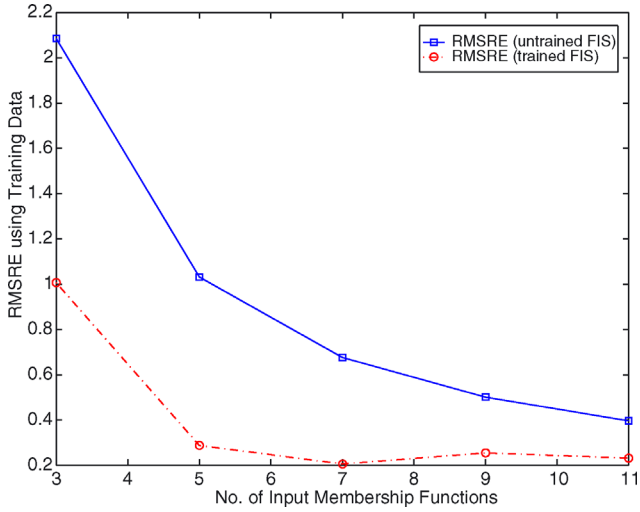

Fig. 13. The RMSRE of the untrained and trained FIS using different number of MFs.

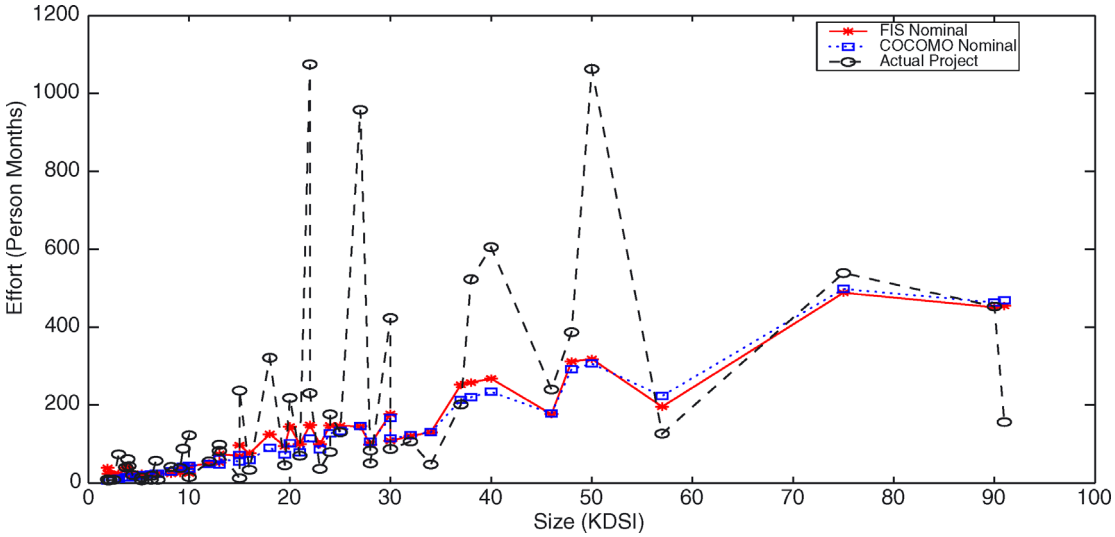

Fig. 14. Nominal effort prediction of trained FIS and COCOMO model on COCOMO database.

Ref. [33], but limit our discussion here to the most accurate training and adaptation strategy. The training algorithms have linear time complexities and run for seconds or minutes in the worst case.

The artificial dataset for all the experiments is composed of 100 observations partitioned into 67 training data and 33 validation data.

5.1. Training the FIS for the framework

We conducted an experiment to train a FIS with five input MFs for the size input variable and three MFs for mode input variable. A plot of the root mean square relative error (RMSRE) showing the learning curve during training is shown in Fig. 10.

The prediction capabilities of the trained FIS was tested using the actual data on which the FIS was trained and on the validation data. Fig. 11 shows the prediction results obtained from the trained FIS using the validation dataset that was not used during training.

Fig. 12 shows the detailed results of instances of the same experiments, using 3, 5, 7, 9, and 11 fuzzy sets for size, respectively. The graph shows consistent improvement in the performance of the trained FIS with increasing number of input MFs when compared to untrained FIS. When 11 MFs were used, the trained and untrained FIS

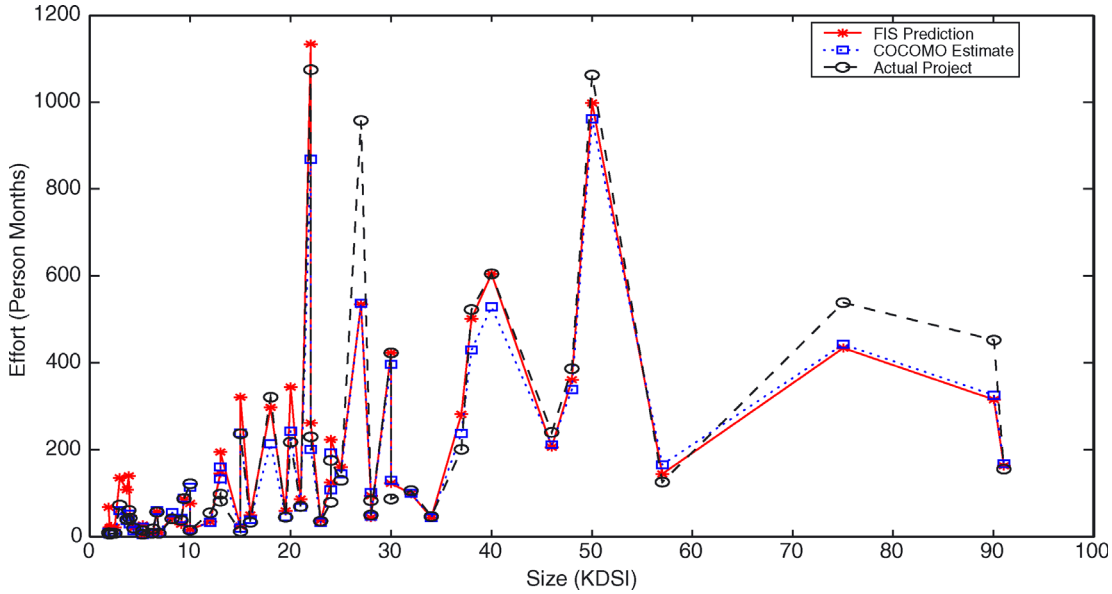

Fig. 15. Effort Prediction of Trained FIS and COCOMO model adjusted by effort multipliers on COCOMO database.

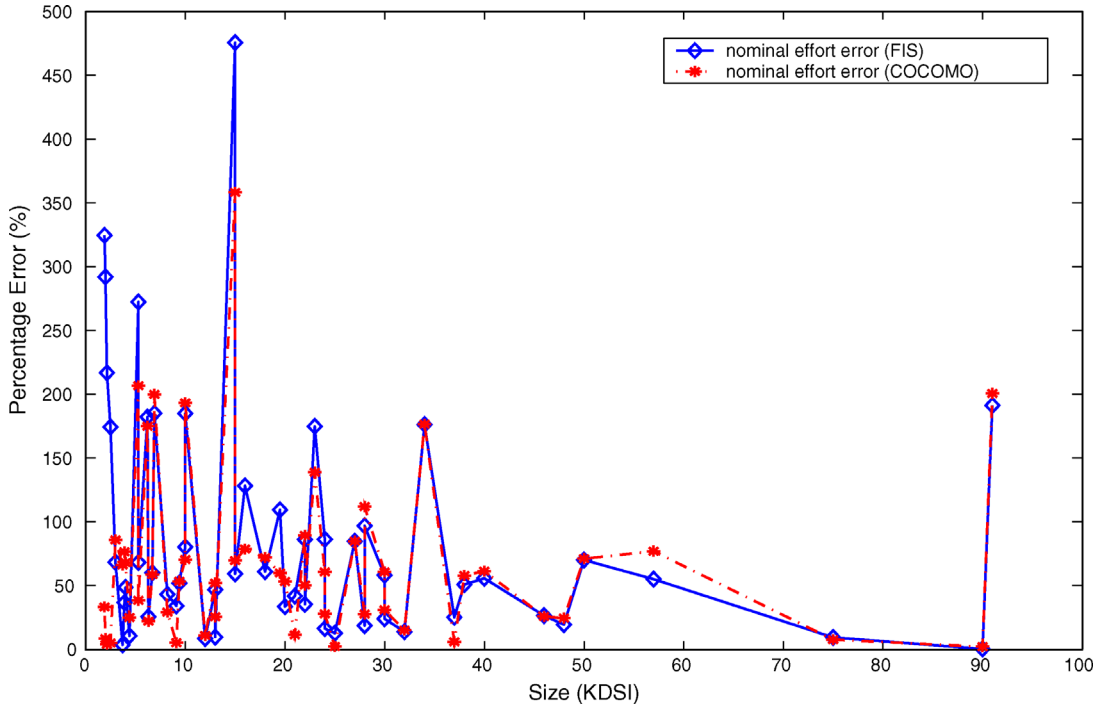

Fig. 16. Percentage error of the nominal effort predictions obtained from trained FIS and COCOMO model using the COCOMO database.

gave the same results. Our intuitive explanation for this is related to the suitability of a rulebase, which depends on the initial fuzzy partitions. That is, if there are too few fuzzy sets, groups of data that should be represented by different rules might be covered by a single rule. In the same vein, if we have too many fuzzy sets than necessary, as is the case for 11 MFs, too many rules would be created resulting in over-fitting. Thus it may be difficult to see significant impact of training on the initial FIS. The accuracy of the trained FIS increases, but interpretability of the rulebase decreases.

In addition to the gain in prediction quality, Fig. 13 shows corresponding reduction in the RMSRE with increasing number of input MFs after training. The gain achieved by training the initial FIS which was hidden when using 11 MFs, as discussed above, is clearly

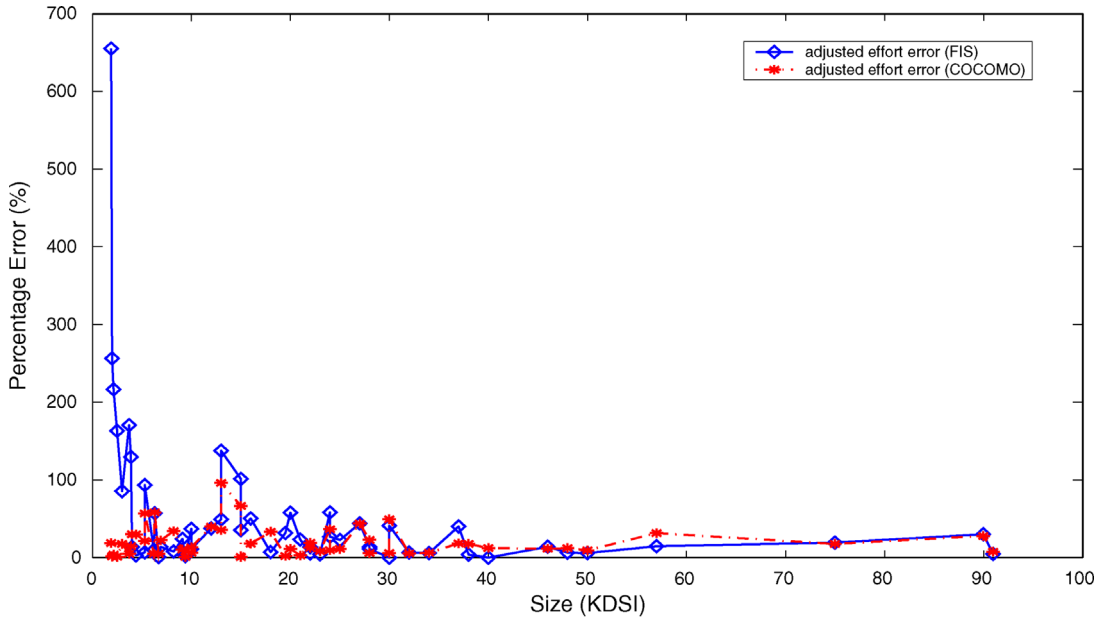

Fig. 17. Percentage error of the adjusted effort predictions obtained from trained FIS and COCOMO model using the COCOMO database with effort multipliers.

revealed in the RMSRE plot. The trained FIS shows lower RMSRE for 11 MFs, meaning that gain was actually recorded in prediction quality, but was over-shadowed by the crisp cut of 25% error margin used. Validation of the training procedure in Section 5.2.3 explains this observation.

5.2. Validation using the COCOMO database

In Section 5.1, the validation of our approach to develop and train FIS for effort prediction has been done using artificial datasets. While real life data has always been adjudged difficult to obtain in software engineering community, the public availability of the COCOMO database offers some respite in this regard.

For the validation in this section, we have selected 53 of the 63 live projects in the COCOMO database [1] whose sizes in lines of code fall within 1–100 KDSI. This is not in anyway a limitation, but it is the same range we have used to develop the FIS from artificial datasets. Our approach to partitioning the input/output space into fuzzy regions discussed in Section 4.3 and Algorithm 1 confirms that we are totally independent of the range of input/output variables.

We adopted two approaches to conduct our validation using the COCOMO database. The first validation approach uses the FIS already trained with artificial training dataset as done in the previous experiments. This validation procedure is presented in Section 5.2.1. Secondly, we trained a new FIS using the data from COCOMO database directly, and testing the performance on the COCOMO data after training as presented in Section 5.2.2.

5.2.1. Using datasets from COCOMO database on trained FIS

Our validation in this subsection is carried out in two steps. The first step compares the performance of the trained FIS for nominal effort prediction with that of nominal intermediate COCOMO model estimate on actual project values. The second step compares the performance of trained FIS with the COCOMO model estimates using the same effort multipliers used in the COCOMO database to adjust the FIS nominal effort. We again compare the performance of adjusted effort of both COCOMO model and trained FIS.

The nominal effort from intermediate COCOMO model has 26% of its predicted values fall within 25% of the actual values (PRED(25) = 26%) on the COCOMO database, while our trained FIS has 23% of its predicted values fall within 25% of the actual values (PRED(25) = 23%). A graph comparing the nominal effort predicted using trained FIS and COCOMO model on actual projects in the database is given in Fig. 14.

In the second validation experiments, tuning with effort adjustment factor (EAF) of the cost drivers in the COCOMO database, the intermediate COCOMO model has 72% of the predicted values fall within 25% of their actual values (PRED(25) = 72%). The trained FIS when tuned with EAF has 55% of the predicted values fall within 25% of their actual values (PRED(25) = 55%). The graph in Fig. 15 compares the adjusted effort predicted using the trained FIS and COCOMO model on actual projects in the database.

We plot a graph of the percentage error on predictions made by trained FIS and COCOMO model using the COCOMO database. These graphs are shown in Figs. 16 and 17 for the nominal effort and adjusted effort predictions, respectively.

Considering the percentage error graphs, the trained FIS recorded marginally higher percentage errors on an average, thus making the predictions of the COCOMO model a little better. The overshoot in the percentage error of one of the data-points when testing the trained FIS could possibly have resulted from the artificial training data not covering values at such data-points.

In summary, the prediction quality and the percentage errors show that COCOMO model marginally outperforms the trained FIS in this test. This observation is not surprising since our trained FIS derived most of its initial knowledge from the COCOMO model, including the training data. For a fairer comparison, we train the FIS directly from the COCOMO database and compare the results as discussed in Section 5.2.2.

5.2.2. Validation using the COCOMO database to train the FIS

The aim of the first validation experiment carried out here is to strengthen our assertion that the availability of more information would enhance the prediction capabilities of the framework.

Two validation experiments were carried out here. The first experiment partitioned the 53 data-points selected from the COCOMO database into 39 training and 14 validation datasets. After training, the FIS yields nominal effort with prediction accuracy of PRED(25) = 26% on training data, PRED(25) = 29% on validation dataset, and PRED(25) = 26% on the entire 53 data-points. This shows that both trained FIS and COCOMO model achieved the same prediction quality—PRED(25) = 26%.

In the second validation experiment, the whole dataset of 53 data-points was used for training, yielding prediction accuracy of PRED(25) = 30% after training. The prediction accuracy recorded by the trained FIS is higher than that of COCOMO model which reported PRED(25) = 26%, as discussed in Section 5.2.1. We are aware that any training procedure that uses same data for training and validation would definitely be biased. But the second experiment was simply carried out to show that, given enough information the framework could perform better.

Fig. 18 shows the percentage prediction error recorded by the trained FIS and COCOMO model when the FIS is trained using all the data-points selected from the COCOMO database. The error graph reveals that, the percentage errors

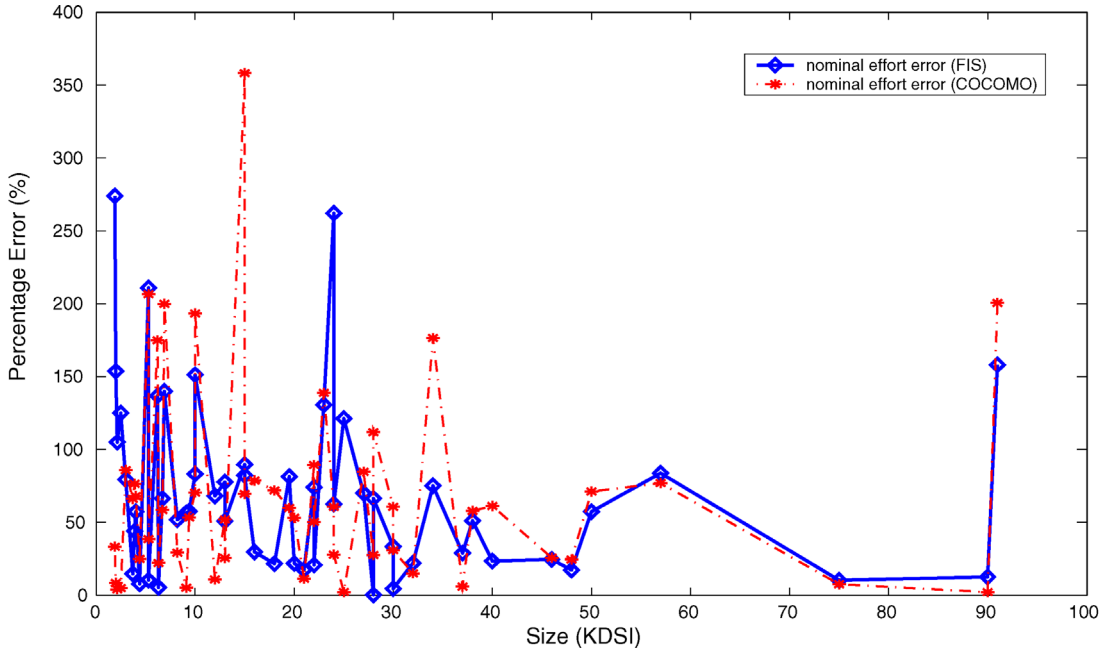

Fig. 18. Percentage error of the nominal effort predictions obtained from trained FIS and COCOMO model using the COCOMO database for training and testing.

of the two are comparable. In sort, the COCOMO model prediction contains the data-point with the largest percentage error.

In concluding our validation using live project data, the FIS appears to perform as well as the COCOMO model, and could potentially perform better, but this needs further extensive empirical investigation overtime.

5.2.3. Validation of the training procedure

The objective of this experiment is to investigate the learning capabilities of the training procedure.

The approach involves developing a badly formulated FIS rulebase and investigates if our training procedure can still rediscover a pattern for improvement. If this objective is realized, then the presence of experts to provide adequate knowledge for building correct rulebase may enhance the potential of a trained FIS in providing qualitative estimates always.

The procedure we took to develop a bad rulebase is to distort the rulebase already well formulated to make it less meaningful. We have neither imposed lower nor upper bounds on the number of rules changed. The objective, as explained, is just to investigate the ability of the training procedure to learn, even when lacking some information. Our procedure for achieving this is given in Algorithm 4.

After distorting the rulebase, it is expected that the prediction accuracy drops, and this actually happened for the FIS with distorted rulebase. We then try to optimize the distorted rulebase to investigate whether our training procedure would be able to make some improvements.

After training, we observed some improvements. In particular, the prediction quality—PRED(25) on testing with the validation data improved from 59.94 to 70.59%, and improves from 54.55 to 62.12% on the training data.

In addition, the training has offered considerable reduction in the RMSRE while testing the trained FIS using validation and training datasets. The importance of this little discovery is again significant. It implies that, while about 70.59% predicted values fall within 25% of the actual values, it is clear that the bulk of the other data-points are very close to the 25% cut, unlike the untrained distorted rulebase FIS that has larger RMSRE.

Fig. 19 shows the percentage error when testing the untrained FIS and trained FIS with distorted rulebase using the training dataset. From the graph, it can be seen that almost 100% of the prediction made by our trained FIS fall within 100% of the actual values while the corresponding untrained FIS has 100% of the prediction fall within 700% of the actual value. The prediction error margin of

Algorithm 4

1. Create a fuzzy inference system with a rulebase formulated using the knowledge of relationships between mode, size and effort in the COCOMO model
2. Extract the rules automatically generated in (1) and save the initial FIS
3. Disorganize the rulebase such that they are less meaningful. For example, a combination of organic mode $M1$, and specific size $S2$ should give effort $C1$. Change the conclusion of this particular rule to $C2$, which is supposed to be the effort for a mode $M1$ and size $S2$
4. Repeat steps 1–3 until some of the rules are made less meaningful based on the relationships in (1)
5. Update the initial FIS with the updated rulebase and save as FIS2
6. Train the new FIS2 to give a finally trained FIS3
7. Evaluate the prediction quality of the trained FIS3, and compare the result of the prediction using the saved FIS2 (5)

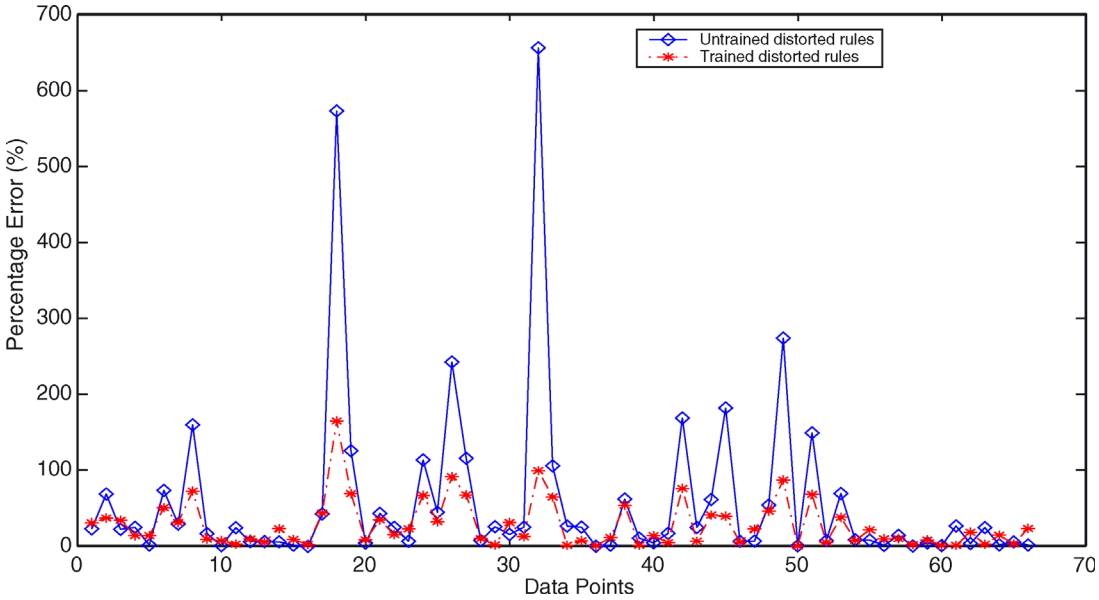

Fig. 19. Percentage error of the predictions obtained using the trained and untrained FIS with distorted rules on TRAINING DATA.

100–700% is too wide, even as the trained FIS still maintains prediction quality that is very much higher than the untrained FIS.

Similarly, Fig. 20 shows similar experimental result on the validation dataset that was not used during training. Again, our trained FIS gave almost 100% of predicted values within 80% error margin, while the untrained FIS gave the same 100% prediction at an error margin of about 400%.

The ability of our training procedure to optimize a badly formulated rulebase with blind knowledge is significant. The availability of experts to furnish the rulebase with meaningful knowledge could make the training procedure a promising candidate for reliable effort prediction.

In summary, we recorded improvements in the performance of our training procedure in the presence of counter-intuitive rulebase. We also witnessed reduction in RMSRE values during validation.

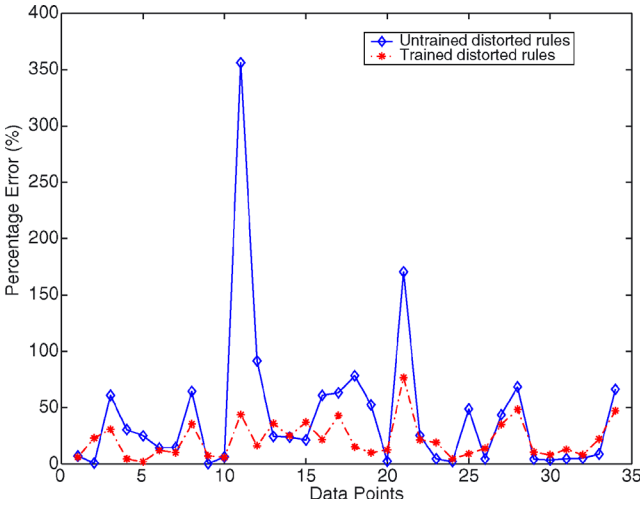

Fig. 20. Percentage error using the trained and untrained FIS with distorted rules on validation data.

6. Conclusion

In this paper, we have presented a transparent FL-based framework, equipped with training and adaptation algorithms for development effort prediction. The framework allows contribution from experts, and also enables the prediction technique to model and adapt to the environment of the prediction problem.

We have demonstrated the capabilities of the framework through empirical validation carried out on artificial datasets and the COCOMO public database of completed projects.

We have reported promising experimental summary results in spite of the little background knowledge in the rulebase and training data. It does signify that there are potentials for improvements when the framework is deployed in practice, since experienced experts could augment with their knowledge.

7. Future research

Some of the open issues, which we have identified, and that can be investigated in future research include the following:

1. Deploying the framework to COCOMO II environment and comparing performance on live projects data. Our framework is still valid and applicable to COCOMO II once experts are available to give information required for MFs definition and rulebase development.

2. Investigating other applicable MF types like Gaussian, and performing comparative analysis can be pursued as a future research.
3. A future research may involve investigating other approaches to normalizing the error measure. In our implementation of the training and adaptation algorithms, we have just used our intuition to normalize so far. Although, we observed improved performance, but require further investigation.
4. Investigating the performance of other defuzzification techniques may be pursued in a future work. The heuristic we have used in modifying the output fuzzy sets considers the defuziffication procedures. In our current implementation, we have used COA and it performed well.
5. Our research presented in this paper has dealt with knowledge imperfection when making decisions about effort and schedule estimations, where knowledge of the software engineer making the estimates can be imperfect due to doubt in validity of knowledge (uncertainty) and/or difficulty in expressing the knowledge (impreciseness). Our research focus for further studies will look at these two types of imperfections in relation to unknown uncertainties that will arise during the development or deployment of the software system. An example of such unknown uncertainties includes the use of untried new technologies or activities that may arise because of unexpected failures or requirements that are encountered.
6. Investigating industrial acceptability of our proposed attributes sets as a groundwork for prediction models evaluation. This could be in the form of metrics based on some or all the attributes.
7. While it is expected that our fuzzy-based technique would be acceptable in the industry, we hope to carry out a feasibility study to establish it.
8. Building a graphical user interface that allows project managers to select input areas in the form of fuzzy sets, rather than forcing them to make crisp commitments.

Acknowledgements

The authors wish to acknowledge King Fahd University of Petroleum and Minerals (KFUPM) for utilizing the various facilities in carrying out this research. We also like to thank Prof. David Rine of George Mason University, Virginia, for his valuable comments. Many thanks are due to the anonymous referees for their detailed and helpful comments.

References

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

[2] B. Boehm, C. Abts, S. Chulani, Software development cost estimation approaches—a survey, Technical Reports, USC-CSE-2000-505, University of Southern California Center for Software Engineering, 2000.

[3] G.D. Boetticher, An assessment of metric contribution in the construction of a neural network-based effort estimator, Proceedings of Second International Workshop on Soft Computing Applied to Software Engineering, 2001.

[4] G.D. Boetticher, Using machine learning to predict project effort: empirical case studies in data-starved domains, Proceedings of Model Based Requirements Workshop, San Diego, 2001, pp. 17–24.

[5] L.C. Briand, I. Wieczorek, Resource estimation in software engineering, in: J. Marciniak (Ed.), Encyclopedia of Software Engineering, second ed., Wiley, New York, 2001.

[6] C.J. Burgess, M. Lefley, Can genetic programming improve software effort estimation? A comparative evaluation, Information and Software Technology 43 (2001) 863–873.

[7] S. Chulani, B. Boehm, B. Steece, Calibrating software cost models using bayesian analysis, Technical Reports, USC-CSE-98-508, University of Southern California Center for Software Engineering, 1998.

[8] B.K. Clark, The Effects of Software Process Maturity on Software Development Effort, PhD Dissertation, Faculty of Graduate School, University of Southern California, Aug 1997.

[9] S. Devnani-Chulani, Bayesian Analysis of Software Cost and Quality Models, PhD Dissertation, Faculty of Graduate School, University of Southern California, May 1999.

[10] Z. Fei, X. Liu, f-COCOMO: fuzzy constructive cost model in software engineering, Proceedings of the IEEE International Conference on Fuzzy Systems, IEEE Press, New York, 1992, pp. 331–337.

[11] N.E. Fenton, S.L. Pfleeger, Software Metrics—A Rigorous and Practical Approach, second ed., PWS Publishing, Boston, MA, 1997.

[12] A.S. Gray, S.G. MacDonell, A comparison of techniques for developing predictive models of software metrics, Information and Software Technology 39 (1997) 425–437.

[13] A.C. Hodgkinson, P.W. Garratt, A neurofuzzy cost estimator, in: Proceedings of the Third International Conference on Software Engineering and Applications—SAE, 1999, pp. 401–406.

[14] A. Idri, A. Abran, COCOMO cost model using fuzzy logic, Seventh International Conference on Fuzzy Theory and Technology, Atlantic City, NJ, 2000.

[15] A. Idri, A. Abran, Evaluating software project similarity by using linguistic quantifier guided aggregations, Proceedings of IFSA/NAFIPS, Vancouver, Canada, 2001, pp. 470–475.

[16] A. Idri, A. Abran, Towards a fuzzy logic based measures for software projects similarity, MCSEAI'2000, Morocco, 2000.

[17] C. Kirsopp, M.J. Shepperd, Making inferences with small numbers of training sets, Sixth International Conference on Empirical Assessment & Evaluation in Software Engineering, Keele University, Staffordshire, UK, April 8th–10th, 2002.

[18] C. Kirsopp, M.J. Shepperd, J. Hart, Search heuristics, case-based reasoning and software project effort prediction, Genetic and Evolutionary Computation Conference (GECCO 2002), New York, AAAI, 2002.

[19] S.G. MacDonell, Software source code sizing using fuzzy logic modeling, Information and Software Technology 45 (2003) 389–404.

[20] S.G. MacDonell, A.R. Gray, A comparison of modeling techniques for software development effort prediction, in: Proceedings of the International Conference on Neural Information Processing and Intelligent Information Systems, Dunedin, New Zealand, Springer, Berlin, 1997, pp. 869–872.

[21] S.G. MacDonell, A.R. Gray, M.J. Calvert, FULSOME: a fuzzy logic modeling tool for software metricians, in: Proceedings of the 18th International Conference of the North American Fuzzy Information Processing Society—NAFIPS, IEEE Press, New York, 1999, pp. 263–267.

[22] S.G. MacDonell, A.R. Gray, M.J. Calvert, FULSOME: a fuzzy logic for software metric practitioners and researchers, in: Proceedings of the Sixth International Conference on Neural Information Processing—ICONIP, IEEE Press, New York, 1999, pp. 308–313.
[23] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, S. Webster, An investigation of machine learning based prediction systems, Journal of Systems and Software 53 (2000) 23–29.
[24] P. Musilek, W. Pedrycz, G. Succi, M. Reformat, Software cost estimation with fuzzy models, Applied Computing Review 8 (2) (2000) 24–29.
[25] D. Nauck, F. Klawonn, R. Kruse, Foundations of Neuro-Fuzzy Systems, Wiley, Chichester, 1997.
[26] D. Nauck, A fuzzy perceptron as a generic model for neuro-fuzzy approaches, in: Proceedings of Fuzzy-Systeme'94, Second GI-Workshop, Munich, Semen Corporation, 1994.
[27] D. Nauck, Data Analysis with Neuro-Fuzzy Methods, Habilitation Thesis, University of Magdeburg, 2000.
[28] M. Negnevitsky, Artificial Intelligence—A Guide to Intelligent Systems, First ed., Addison-Wesley, Reading, MA, 2002.
[29] W. Pedrycz, H.F. Peters, S. Ramanna, A fuzzy set approach to cost estimation of software projects, Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, Alberta, Canada, May 9–12 (1999).
[30] S.L. Pfleeger, Model of software effort and productivity, Information and Software Technology 33 (3) (1991) 224–231.
[31] J. Ryder, Fuzzy modeling of software effort prediction, Proceedings of IEEE Information Technology Conference, Syracuse, NY, 1998.
[32] M.O. Saliu, M. Ahmed, Soft computing based effort prediction systems—A survey, in: E. Damiani, L.C. Jain (Eds.), Computational Intelligence in Software Engineering, Springer-Verlag, July 2004, ISBN 3-540-22030-5.
[33] M.O. Saliu, Adaptive Fuzzy Logic Based Framework for Software Development Effort Prediction, MS Thesis, Deanship of Graduate Studies, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, April 2003.
[34] C. Schofield, Non-algorithmic effort estimation techniques, Technical Reports, Department of Computing, Bournemouth University, England, TR98-01, March 1998.
[35] M. Shepperd, G. Kadoda, Comparing software prediction techniques using simulation, IEEE Transactions on Software Engineering 27 (11) (2001) 1014–1022.
[36] Y. Shi, M. Mizumoto, N. Yubazaki, M. Otani, A learning algorithm for tuning fuzzy rules based on the gradient descent method, Proceedins of Fifth IEEE International Conference on Fuzzy System, New Orleans, Sept. 8–11, 1996, pp. 55–61.
[37] K.K. Shukla, Neuro-Genetic prediction of software development effort, Information and Software Technology Journal 42 (2000) 701–713.
[38] K. Srinivasan, D. Fisher, Machine learning approaches to estimating software development effort, IEEE Transactions on Software Engineering 21 (2) (1995).
[39] K. Strike, K. El-Emam, N. Madhavji, Software cost estimation with incomplete data, IEEE Transactions on Software Engineering 27 (10) (2001).
[40] A.R. Venkatachalam, Software cost estimation using artificial neural networks, in: Proceedings of the International Joint Conference on Neural Networks, 1993, pp. 987–990.
[41] L.-X. Wang, J.M. Mendel, Generating fuzzy rules by learning from examples, IEEE Transactions on System, Man, and Cybernetics 22 (6) (1992).
[42] G. Wittig, G. Finnie, Estimating software development effort with connectionist models, Information and Software Technology 39 (1997) 469–476.