

2013

# EEF-CAS: An Effort Estimation Framework with Customizable Attribute Selection

Katarina Grolinger  
kgroling@uwo.ca

Besa Muslimi  
bmuslimi@alumni.uwo.ca

Miriam A.M. Capretz  
Western University, mcapretz@uwo.ca

Mark Benko  
markbenko@hotmail.com

Follow this and additional works at: <https://ir.lib.uwo.ca/electricalpub>

 Part of the [Software Engineering Commons](#)

---

## Citation of this paper:

Grolinger, Katarina; Muslimi, Besa; Capretz, Miriam A.M.; and Benko, Mark, "EEF-CAS: An Effort Estimation Framework with Customizable Attribute Selection" (2013). *Electrical and Computer Engineering Publications*. 24.  
<https://ir.lib.uwo.ca/electricalpub/24>

## EEF-CAS: An Effort Estimation Framework with Customizable Attribute Selection

Besa Muslimi, \*Katarina Grolinger, Miriam A.M. Capretz, Mark Benko  
*Department of Electrical and Computer Engineering, Faculty of Engineering, Western  
University, London, Canada*  
*bmuslimi@alumni.uwo.ca, {kgroling, mcapretz}@uwo.ca, markbenko@hotmail.com*  
*\*Corresponding Author*

### Abstract

*Existing estimation frameworks generally provide one-size-fits-all solutions that fail to produce accurate estimates in most environments. Research has shown that the accomplishment of accurate effort estimates is a long-term process that, above all, requires the extensive collection of effort estimation data by each organization. Collected data is generally characterized by a set of attributes that are believed to affect the development effort. The attributes that most affect development effort vary widely depending on the type of product being developed and the environment in which it is being developed. Thus, any new estimation framework must offer the flexibility of customizable attribute selection. Moreover, such attributes could provide the ability to incorporate empirical evidence and expert judgment into the effort estimation framework. Finally, because software is virtual and therefore intangible, the most important software metrics are notorious for being subjective according to the experience of the estimator. Consequently, a measurement and inference system that is robust to subjectivity and uncertainty must be in place. The Effort Estimation Framework with Customizable Attribute Selection (EEF-CAS) presented in this paper has been designed with the above requirements in mind. It is accompanied with four preparation process steps that allow for any organization implementing it to establish an estimation process. This estimation process facilitates data collection, framework customization to the organization's needs, its calibration with the organization's data, and the capability of continual improvement. The proposed framework described in this paper was validated in a real software development organization.*

**Keywords:** *Software Development Effort Estimation, Customizable Attribute Selection  
Expert Judgment, Empirical Evidence, Neuro-Fuzzy Inference*

### 1. Introduction

Software effort estimation has attracted, and continues to attract, significant research attention. In 2007, Jørgensen and Shepperd [1] reviewed 304 software estimation papers from 76 journals. Even though this survey did not include research published in conference proceedings, it illustrated the great challenge of software effort estimation as well as its complexity and significance.

This considerable attention has resulted from the need for accurate effort estimates for contract bidding, project planning and control, budgeting, and risk analysis. Based on these estimates, key project decisions are made, feasible performance objectives are defined, and schedules are set up. Overestimation leads to lost bids for projects, while underestimation leads to runaway projects and unsatisfied customers. Unfortunately, software effort estimation still remains fairly inaccurate: on average, software projects expend thirty to forty percent more effort than estimated [2]. The reasons for this are numerous and include intrinsic software complexity, uncertainties in the software development process, dynamic and interdependent product and process variables, lack of software data, and diversity of software products and development methods.

Yet, as the software industry continues to expand in wide-ranging and far-reaching directions, its products becoming vital components of every other industry in the world, it is important that accurate estimates no longer be perceived as luxuries but as essential information to the business of software development. Thus, there exists a need for a reliable software development effort estimation framework.

Even though the majority of software estimation research focuses on formal models [1], there is no conclusive evidence that formal models produce more accurate estimates than expert estimation [3]. An

approach is considered an expert estimation if the final quantification step which produces the actual estimate is a judgmental process; an expert assesses the effort based on available information. Nevertheless, intuition is part of the estimation process. On the other hand, if the final quantification step is mechanical, the approach is categorized as a formal model. Furthermore, Jørgensen's survey [3] found that expert estimation produced more accurate estimates in ten of the sixteen studies reviewed. Moreover, expert estimation approaches are, by large, the most commonly used approaches in industry [1, 2].

Fenton and Neil [4] developed a list of the desired characteristics that a software development effort estimation framework must have. The characteristics they list are:

- The ability to handle diverse process and product variables
- The ability to incorporate empirical evidence and expert judgment
- The ability to determine genuine cause and effect relationships
- The ability to handle uncertainty
- The ability to handle incomplete information

Other studies have identified further gaps in current software estimation research, such as the need to explore estimation methods in real-life estimation situations [1], the need to use newer data sets [1], the importance of feature selection [5], and the need for "white box" approaches which can explain cause-effect relationships [4, 6].

In response to these concerns, the Effort Estimation Framework with Customizable Attribute Selection (EEF-CAS) is proposed in this research. In the EEF-CAS attribute selection step, the organization selects the factors that it believes most influence the development task effort. These factors include any feature that can affect the amount of effort required, including system characteristics and personal skills. Therefore, EEF-CAS can accommodate different systems and personal skills.

Multilayer feedforward neural networks are used to model the relationship between development effort and the factors that affect it, while fuzzy logic is incorporated to deal with the uncertainty and subjectivity present in these factors. And finally, rules (which can be validated by experts) are extracted from a trained neural network and embedded into the Adaptive Neuro-Fuzzy Inference System (ANFIS). The Effort Estimation Framework with Customizable Attribute Selection (EEF-CAS) thus should be able to handle diverse, organization-specific variables in its attribute selection stage, incorporate empirical evidence using neural networks, handle imprecision using fuzzy logic, and establish cause-effect relationships through rule extraction from a neural network

The remainder of the paper is organized as follows: related work is reviewed in Section 2, while Section 3 portrays the proposed EEF-CAS. Section 4 depicts a case study; limitations of this work are presented in Section 5; and contributions and conclusions are drawn in Section 6.

## 2. Related work

There are three different categories of studies related to this research. The first category, described in Section 2.1, covers existing frameworks for software effort estimation. The second category, presented in Section 2.2, consists of studies which use a variety of soft computing approaches for effort estimation. The third category, described in Section 2.3, contains studies that deal with identification of influential factors for software effort estimation.

### 2.1. Frameworks for software effort estimation

A number of frameworks for software effort estimation have been proposed by various authors, including Shukla and Misra [8], Pendharkar and Rodger [9], Sharma and Verma [10], Ahmed *et al.* [11], and Huang *et al.* [12]. The work of Shukla and Misra [8] focused on effort estimation for software maintenance, while the other frameworks, as well as the proposed EEF-CAS, deal with estimation for software development. Pendharkar and Rodger [9] used a distributed problem-solving approach. Their software effort estimation model is based on four specific project attributes: team size, software size, language type, and CASE tool. The proposed EEF-CAS, however, is more flexible and allows the estimator to choose the project attributes.

Similar to this study, the three frameworks proposed by Sharma and Verma [10], Ahmed *et al.* [11], and Huang *et al.* [12] also used fuzzy logic. However, in contrast to this work, all three studies were based on the COCOMO model. Sharma and Verma [10] extended COCOMO by incorporating fuzziness into size measurement, mode of development, and cost drivers. Ahmed *et al.* [11] proposed an adaptive fuzzy logic-based framework which consisted of two parts corresponding to the two parts of the COCOMO intermediate model: fuzzy COCOMO for nominal effort and fuzzy adjustment factors to fuzzify the COCOMO cost drivers. Huang *et al.* [12] used COCOMO as the algorithmic model within their framework; nevertheless, this default algorithmic model can be replaced by another. Another similarity between the work of Huang *et al.* [12] and this study is the use of an adaptive neuro-fuzzy inference system (ANFIS). However, while Huang *et al.* used ANFIS to prepare input variables for the algorithmic model, we have used it in the final, optional model step for fine-tuning of fuzzy sets.

## 2.2. Soft computing approaches for effort estimation

Soft computing approaches to estimate effort are numerous; therefore, only a few representative studies are discussed here. Boetticher [13] used a multilayer backpropagation network to predict actual effort. Although this approach contains some novel ideas, the quantitative inputs must be known at a level of detail that is often impossible at the time estimates are created. Finnie and Wittig [14] also conducted experiments using neural networks. Both these studies showed that for estimation using neural networks, having more input attributes than the estimated size of the software yields more accurate results.

In an attempt to build on the success of neural networks while avoiding the reasoning opacity that accompanies their use, Huang used neuro-fuzzy logic to estimate software development effort [15]. He did so by fuzzifying the inputs of the COCOMO model and then used them with data to train a neuro-fuzzy system. Although the resulting neuro-fuzzy model outperformed COCOMO, its potential for further improvement is questionable because the model uses the COCOMO regression equation instead of rules to infer estimates. By doing so, the model inherits the limitations of the COCOMO regression technique.

Idri and Elyassami [6] proposed the Fuzzy ID3 Decision Tree, which integrates ID3 decision trees with fuzzy logic. The decision-tree technique is a white-box approach which can provide explanations of cause-effect relationships while fuzzy logic handles uncertain and imprecise data. Azzeh *et al.* [7] proposed Fuzzy Grey Relational Analysis, which combined fuzzy set theory with Grey Relational Analysis. Bhatnagar *et al.* [16] provided a review of software effort estimation studies based on fuzzy logic. Even though this review is not comprehensive, clearly a number of studies have built upon the COCOMO model. By contrast, COCOMO does not form part of the EEF-CAS approach.

## 2.3. Factors influencing software effort estimation

The number and type of factors that are considered influential differ among estimation models and techniques. In practice, such factors vary greatly depending on the development environment and the type of system being built. However, the factors that are often found to have a strong effect on software development effort are expert judgment, task implementer capability, and complexity.

**Expert judgment:** Jørgensen [17] summarized a number of studies on expert estimation, including how often it has been used in the software development industry, why it is preferred, and how well it performs compared to other estimation models. This study revealed that informal expert estimation is the most widely used estimation technique in companies all over the world. Although the results show that no existing effort estimation model is very accurate, they also show that experts are useful resources for estimation. In fact, most industry and academic researchers agree that an expert's opinion is not only useful but often necessary when making estimates [18–20]. Consequently, expert judgment is taken into account by giving the expert the ability to select the attributes that are believed to most influence software development task effort by the organization implementing the EEF-CAS.

**Task implementer capability:** another important factor that influences the estimated effort is the quality and capability of the task implementer [18, 19, 21]. Therefore, any estimation model must include implementer capability as one of its inputs. However, this can be challenging when software effort is being determined for a more granular task and the identity of the task implementer is known, because organizations can then be reluctant to evaluate the capability of that implementer. This

reluctance is due to the existing confidentiality agreement between employer and employee, which such an evaluation could breach, and to the decrease in morale that a low evaluation would produce. Therefore, in some organizations, it may be impossible to include implementer capability in the evaluation. In this case, other inputs can be included that evaluate the implementer's level of familiarity with the technology, functionality, language, and domain associated with the task. Hence, in the proposed EEF-CAS, the estimator can select such profiling attributes.

**Complexity:** this is another attribute that is considered to have a strong effect on software development effort [19–21]. Glass [18] has noted that for every 25% increase in problem complexity, there is a 100% increase in the complexity of the software solution. Moreover, managers consider complexity to be the most significant factor in project estimation [21].

Khalidi *et al.* [22] performed a regression analysis over six public cost estimation data sets to select a set of attributes which strongly affect the effort estimate. For each data set, they determined the most effective attributes. Next, they compared the results and proposed a standard set of metrics. The flexibility of the proposed EEF-CAS enables the estimator to select the profiling attributes that they believe influence their software development task effort.

### 3. The effort estimation framework with customizable attribute selection

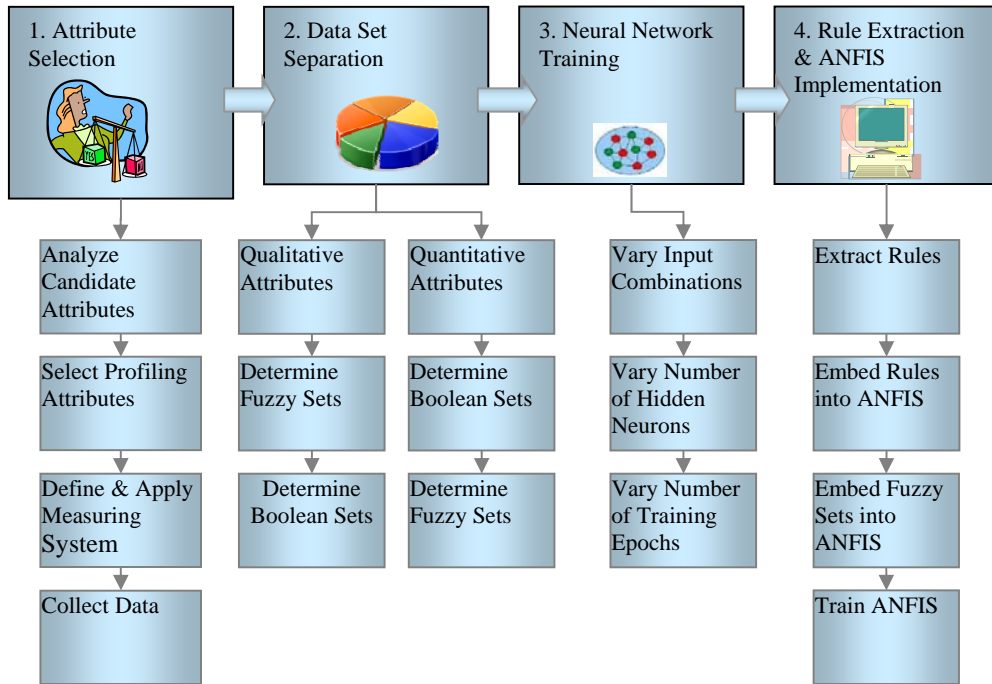
The Effort Estimation Framework with Customizable Attribute Selection (EEF-CAS) has been designed to include several desired characteristics which are lacking in existing estimation frameworks. Specifically, EEF-CAS offers the following features:

- Diverse variables are handled in the attribute selection stage when profiling attributes are chosen by the estimator.
- Empirical evidence is incorporated through the use of historical data for neural network training.
- Imprecision and uncertainties are handled using fuzzy logic inference.
- Cause-effect relationships are determined by rule extraction from the neural network.

Fig. 1 depicts the EEF-CAS preparation process. As shown, there are four preparation steps that must be completed before the EEF-CAS is closely customized to the organization's environment and ready for use:

1. Attribute selection
2. Data set separation for
  - a. Qualitative attributes
  - b. Quantitative attributes
3. Neural network training
4. Rule extraction and ANFIS implementation.

In the first step, qualitative and quantitative profiling attributes that are believed to most influence software development task effort are selected by the organization implementing the EEF-CAS. These attributes are defined and a measuring system is applied to each of them, allowing them to function as metrics. These attributes then serve as inputs to the EEF-CAS and each software development task is profiled with them. The profile, together with the actual effort required to complete the task (known once the task has been completed) are considered as one data point. When a sufficient amount of data points have been collected, the second step of the preparation process begins: The values of each attribute are separated into fuzzy and Boolean sets. The Boolean sets allow the attribute values to be transformed into Boolean data. The fuzzy sets are not used until Step 4 of the preparation process. The third step consists of using the Boolean data to train neural networks where the output of the neural network is estimated effort. In the fourth and final step, rules that model the relationship between the profiling attributes and development effort are extracted from the most successfully trained neural network of Step 3. They are then embedded into an adaptive neuro-fuzzy inference system, as are the fuzzy sets determined in Step 2. Finally, the historical data collected is used, once again, to train the ANFIS and fine-tune the fuzzy sets. At this point, the EEF-CAS is calibrated to the organization's environment and ready to be used to estimate future software development tasks.



**Figure 1.** The Effort Estimation Framework with Customizable Attribute Selection (EEF-CAS)

### 3.1. Step 1: Attribute selection

A *profiling attribute* is a measurable system characteristic, personal skill, or anything else that can affect the amount of effort required to complete a software development task.

Research has shown that of the hundreds of parameters which can affect software development effort, only a few may influence productivity in a given environment [23]. Therefore, the first EEF-CAS step allows the estimator to select the factors that it believes most influence development effort, given their product and the environment in which they operate. To enable each organization to select the few attributes that most affect its productivity, it is recommended that many attributes be initially selected for measurement and recording. Those that turn out to be irrelevant during the neural network training stage, EEF-CAS Step 3, can be discarded, and the most strongly predictive ones can be kept. Ideally, the final number of profiling attributes selected as inputs to the EEF-CAS should be small. The reason for this is that the EEF-CAS uses a neural network to learn input-output relationships from historical data, and the larger the number of input attributes, the larger will be the volume of data needed to train the neural network successfully. In fact, the number of data points needed to train a neural network grows exponentially with additional inputs [24].

The EEF-CAS can accommodate both quantitative and qualitative profiling attributes. Although quantitative attributes can be directly used as numbers, qualitative attributes must be clearly defined to indicate what is being measured.

#### 3.1.1. Defining and applying the measuring system

The EEF-CAS qualitative attribute measuring system was designed to allow the use of qualitative attributes and to enable each such attribute to be measured according to the needs of the organization using it. Such a measuring system requires that each qualitative attribute be defined and then be further decomposed into sub-attributes. Each sub-attribute should be a factor that affects the evaluation of the profiling attribute. Once the user evaluates the sub-attributes, their values can be averaged and used as the value of the overall profiling attribute. To facilitate this process, Tables 1 and 2 were designed and should be completed for each selected qualitative profiling attribute. Table 1 should also be used to define quantitative attributes.

**Table 1.** Format for defining profiling attributes

Name	The name of the attribute and any short names used for it.
Definition	A clear and concise definition that indicates what is being measured by the attribute. All attributes should be defined in terms of how they affect effort.
Rationale	The rationale used to select the attribute. One or more examples may be given to clarify the rationale.
Implementation	A way of combining all the sub-attributes into a single unit of measurement (usually the arithmetic mean). Also, any clarifying notes on how the attribute should be perceived or evaluated.

**Table 2.** Format for defining profiling sub-attributes

Sub-Attribute Name			
Definition	A clear and concise definition of the sub-attribute		
Definitions of Scale Values	Low	The definition of the “Low” scale value for this particular sub-attribute.	An arrow depicting the direction in which the effort estimate varies with increasing sub-attribute value.
	Medium-Low	No definition required.	
	Medium	The definition of the “Medium” scale value for this particular sub-attribute.	
	Medium-High	No definition required.	
	High	The definition of the “High” scale value for this particular sub-attribute.	

The definitions of the scale values required in Table 2 serve as guidelines for the estimator. Each scale value corresponds to a number between one and five, with one corresponding to the *Low* set and five corresponding to the *High* set. The overall profiling attribute value is the arithmetic mean of the values of its sub-attributes.

Although breaking down profiling attributes into sub-attributes does lengthen the process, it is beneficial in the long term because it enables the collection of more accurate data. Qualitative attributes that affect software development effort usually encompass several aspects of the quality they describe. For example, when defining the required reliability of a software system, one could consider how a system failure would affect the environment and the users of the system (i.e., a mere inconvenience versus endangerment of human life), the acceptable frequency of failures (mean time to failure, or MTTF), and the acceptable repair time (mean time to repair, or MTTR). Although all three of these factors affect the reliability of a system, they do so in different ways, often resulting in different evaluations. Therefore, clustering these three factors and measuring them as one would introduce a large amount of uncertainty and inaccuracy into the measurement. On the other hand, decomposing the attribute into sub-attributes makes it possible to collect more accurate data.

Table 3 shows the application of Table 1 to the *Reliability* profiling attribute, and Table 4 shows the application of Table 2 to its sub-attributes.

### 3.1.2. Data collection

After profiling attributes have been chosen and defined and the qualitative data measuring system has been established, data collection can commence. However, the question remains of what is a sufficient number of data points. Research has shown that the number of training data points required for a neural network that contains  $W$  weighted connections is given by Baum and Haussler [25]:




$$m > \frac{W}{\varepsilon},$$

where  $m$  is the number of training data points and  $\varepsilon$  is the allowable fraction of error on the training set. If  $\varepsilon$  is assumed to be less than 0.125, then approximately ten training data points are required for each weighted connection in the neural network. Therefore, assuming that an organization knows the largest network architecture to be used in EEF-CAS Step 3, it must collect at least ten times as many data points as the number of weighted connections within it.

**Table 3.** The *Reliability* profiling attribute

Name	<i>Reliability</i>
Definition	The degree of reliability required from the component or functionality implemented in the task.
Rationale	A task that involves the development of a highly reliable component or functionality generally requires more effort.
Implementation	Each sub-attribute is evaluated as Low, Medium-Low, Medium, Medium-High, or High, corresponding to values between 1 and 5, respectively. The average of the sub-attribute values is the overall attribute value.

**Table 4.** Definition of *Reliability* sub-attributes

<i>Criticality</i>		
Definition	The problem created if the component or functionality implemented in this task fails.	
Definitions of Scale Values	Low	No critical data will be lost.
	Medium	Some business data may be lost causing a day's work set back.
	High	Business data will be lost causing a major set back
		
<i>Mean Time to Failure (MTTF)</i>		
Definition	The degree of importance that the particular component/functionality being implemented should rarely fail.	
Definitions of Scale Values	Low	May fail often (once every few days).
	Medium	May fail between once a month and once in three months.
	High	Should not fail more than once in six months.
		
<i>Mean Time to Repair (MTTR)</i>		
Definition	The degree of importance that the particular component/functionality being implemented in the task have a short repair time.	
Definitions of Scale Values	Low	Not a significant problem if the component/ functionality is down for a week or less.
	Medium	Important that it not be down for more than a day.
	High	Very important that it not be down for more than an hour.
		

### 3.2. Step 2: Data set separation

The second EEF-CAS step is data set separation. To be able to extract rules from a trained neural network using the rule-extraction technique, the data with which the neural network is trained must be Boolean rather than continuous. Subsection 3.2.1 describes the separation of the range of values for qualitative attributes into fuzzy sets and then into Boolean sets, while Subsection 3.2.2 describes the separation of the range of values for quantitative attributes into Boolean sets and then into fuzzy sets. Note that the data set separation process must also be applied to the output attribute, effort.

#### 3.2.1. Data set separation for qualitative attributes

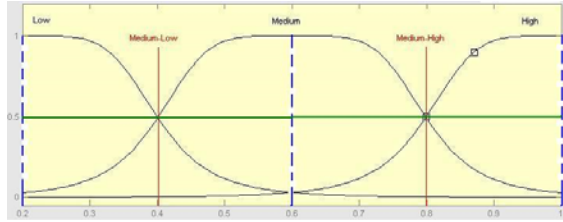
Each qualitative profiling attribute can have a value between one and five, and therefore when these values are normalized, the range of values becomes 0.2 to one. The value zero is reserved to indicate the value “not applicable”.

Next, the boundaries of the fuzzy sets are determined. Each qualitative attribute is separated into three fuzzy sets, *Low*, *Medium*, and *High*, as shown in Fig. 2 [26]. The generalized bell function was selected as the membership function for two reasons: it is nonlinearly smooth, and it offers three adaptable parameters with which the shape of the function can be customized during ANFIS training [27].

The boundaries of the fuzzy sets *Low* and *Medium* are determined so that the crossover point for each set is at the *Medium-Low* value of two, or 0.4 when normalized, as shown in Fig. 2. Likewise, the boundaries of the fuzzy sets *Medium* and *High* can be determined so that the crossover point for each set is at the *Medium-High* value of four, or 0.8 when normalized. The crossover point of a fuzzy set *A* is any point where  $\mu_A(x) = 0.5$ . The rationale for locating the fuzzy set crossover points at these values is simple and straightforward: the points *Medium-Low* and *Medium-High* are designed to be the points halfway between one set and the next one. Therefore, it is only logical that they would be used as the fuzzy-set crossover points. Fig. 2 illustrates this concept by showing that the crossover points of the



fuzzy sets occur at the *Medium-Low* and *Medium-High* values. This condition determines the values for parameters  $a$  and  $b$  of the generalized bell function. In addition, the center of each of the membership functions, the parameter  $c$  of the generalized bell function, is located at the corresponding value that it represents, as illustrated by the dashed lines. For example, the center of the generalized bell function representing fuzzy set *Low* is at 0.2.



$$bell(x; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}} \quad [26]$$

**Figure 2.** Fuzzy sets of qualitative profiling attributes

After the fuzzy sets have been determined, the Boolean sets can be established. The boundaries of the Boolean sets should be located at the crossover points of each fuzzy set. The points at the boundaries should be included in the set that has the smaller range of values included. Therefore, data points with qualitative attribute values of 0.4 should be included in the *Low* set.

Once the boundaries of the Boolean sets have been determined, the data value transformation from continuous to Boolean can proceed. Each continuous value is transformed into a three-element Boolean vector, where one signifies the set to which the value belongs and zero signifies the two sets to which it does not. The Boolean vector's first element represents the *Low* set, its second element represents the *Medium* set, and its third element represents the *High* set, i.e., [*Low Medium High*]. Only one of the three sets must be set to one, and the other two must be set to zero. For example, Table 5 shows a fictional data point profiled by two qualitative attributes. The first row contains attribute values and the overall data point in continuous vector format, [0.633 0.25]. The second row contains the Boolean transformation of the data point shown in the first row. The Attribute 1 value has been transformed into [0 1 0] because the value 0.633 falls within the bounds of the *Medium* set. The Attribute 2 value has been transformed into [1 0 0] because the value 0.25 belongs to the *Low* set.

**Table 5.** Conversion of a continuous data point into Boolean format

Format	Qualitative Attribute 1	Qualitative Attribute 2	Data Point
Continuous	0.633	0.25	[0.633 0.25]
Boolean	[0 1 0]	[1 0 0]	[[0 1 0] [1 0 0]]

### 3.2.2. Data set separation for quantitative attributes

Because the range and distribution of values used for quantitative attributes is organization-dependent, it is challenging to define an exact process for separating quantitative attribute values into sets. However, guidelines are provided here that should be followed when determining these sets.

First and foremost, each Boolean set must contain a sufficient number of data points so that the neural network is able to associate certain input values with that particular set. If only the minimum number of points has been collected (i.e., ten times the number of weighted connections in the largest neural network architecture to be used), the quantitative-attribute Boolean sets can be separated so that each set contains an approximately equal amount of data. This will ensure that each neural-network training set contains enough data for the neural network to learn. If the amount of data collected greatly exceeds the minimum required amount, then other approaches can be used, such as self-organizing maps or clustering algorithms.

In most cases, because data collection takes a long time, most companies will start implementing the second EEF-CAS step once they have gathered what they consider to be sufficient data. In this case, the Boolean set boundaries must be determined based on an equal-data-amount criterion.

Once the Boolean set boundaries have been determined, the fuzzy set membership functions can be defined for the sets of quantitative profiling attributes. The generalized bell function is the membership function of choice. The width of each bell membership function, which is controlled by the parameter  $a$ ,

should be equal to the width of the corresponding Boolean set. Moreover, the center of the generalized membership function of each fuzzy set, which is controlled by the parameter  $c$ , should be located at the median value of the set it represents. For example, to determine where the center of the generalized bell membership function of the fuzzy set *Low* should be located, the median of the data values contained in the Boolean set *Low* should be used. The reason why the median, rather than the mean, was chosen as the center of the generalized bell membership functions was that the finite breakdown point of the median is much higher than that of the mean [28]. The finite breakdown point is the smallest proportion of outliers that can result in the mean or the median being arbitrarily large or small for a given set of observations [28].

Finally, for the quantitative output attribute, *Effort*, no fuzzy sets are required because the zero-order Sugeno inference system implemented in the ANFIS does not require fuzzy sets for the output attribute. Instead, one constant value must be selected to represent each output set. For the same reasons discussed above, the median is the preferred representative value for each output attribute set.

### 3.3. Step 3: Neural network training

This third step of EEF-CAS, neural network training, is performed using Boolean data from the data set separation step. This step involves to some extent trial and error because of the many different factors that can influence how successfully a neural network can be trained. In this section, the three most significant factors will be discussed: the inputs, the number of hidden neurons, and the number of epochs.

The most important factors are the inputs: the stronger the relationship between the inputs and the outputs, the easier it is for the neural network to learn the relationship [29]. To determine the combination of profiling attributes that have the greatest influence on development effort, different combinations of the inputs must be tested. The number of profiling attributes collected and the number of sets defined for each attribute determine the number of inputs to the network. For example, if three profiling attributes are selected, and the values of each one are separated into three sets, *Low*, *Medium*, and *High*, then the neural network will have nine inputs.

The next factor that affects neural network training is the number of neurons used in the middle (hidden) layer. As the number of neurons in the hidden layer increases, so does the accuracy of the neural network in predicting the output of the training data [30]. However, if the number of hidden neurons is too large, the network loses its ability to generalize and models itself too closely to the training data. Consequently, over-fitting occurs, and the network performs well when the training data are used, but poorly when new data are entered [30]. Although several methods have been proposed to determine the number of hidden-layer neurons [30–32], at this stage of this research, a trial-and-error process has been used. Several different networks with varying numbers of hidden neurons were trained and tested, and the architecture which yielded the best results in accurately predicting both training and testing data was chosen.

Finally, the number of training epochs can be varied to determine whether training the neural networks with more epochs yields significantly more accurate results. Although it has been shown that most networks can be successfully trained with 1000 epochs, this number can sometimes vary [33].

### 3.4. Step 4: Rule extraction and ANFIS implementation

The final EEF-CAS step is optional and consists of rule extraction and ANFIS implementation. Upon its completion, the EEF-CAS can be used as a effort estimator for software development tasks. The framework is functional without this step, but its omission results in the inability to establish cause-effect relationships because rule extraction from the neural network will not be performed.

#### 3.4.1. Rule extraction

Rules are extracted from the neural network that was most successfully trained in Step 3. Specifically, the most general rules are extracted from its hidden-layer neurons and its output-layer neurons and then combined to create rules that model the relationship between the profiling attributes and the output effort.

The framework can accommodate different rule extraction techniques, including the decomposition approach in which the focus is on extracting rules at the level of individual hidden and output neurons [34]. Each hidden and output neuron is interpreted as a Boolean rule, and the rules are extracted by finding combinations of incoming weights whose sum exceeds the threshold of the neuron. Specifically, the approach proposed by Krishnan *et al.* [35] has been used here, which provides a simple yet efficient technique for rule extraction from feedforward neural networks with Boolean inputs. By sorting the input weights to a neuron and ordering the weights suitably, Krishnan *et al.* pruned the search space, increasing algorithm efficiency.

### 3.4.2. ANFIS implementation

Neuro-fuzzy systems combine fuzzy inference and neural computing to provide a mechanism that learns from data and is robust to uncertainty and incomplete data. Here, the Adaptive Neuro-Fuzzy Inference System (ANFIS) [36] is used, which combines Sugeno fuzzy inference [29, 37] with a multilayer feedforward neural network [38]. The rule base of a zero-order Sugeno ANFIS system must be known in advance, while the parameters of the membership functions used for the fuzzy sets of the inputs are adjusted during the training of the ANFIS system.

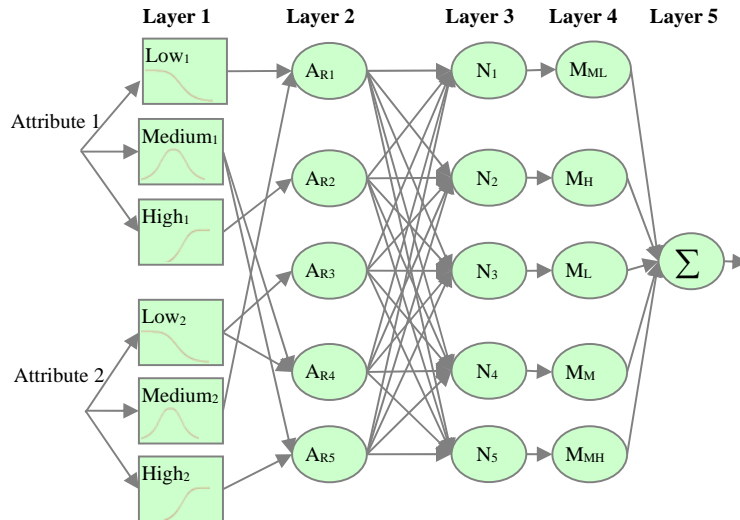
The rules extracted from the neural network in the previous EEF-CAS step are embedded into an ANFIS system. For example, suppose that the following five rules were extracted from the neural network:

- IF Attribute 1 is Low AND Attribute 2 is Medium THEN Output is  $M_{ML}$ .
- IF Attribute 1 is High THEN Output is  $M_H$ .
- IF Attribute 2 is Low THEN Output is  $M_L$ .
- IF Attribute 1 is Medium AND Attribute 2 is Low THEN Output is  $M_M$ .
- IF Attribute 1 is Medium AND Attribute 2 is High THEN Output is  $M_{MH}$ .

$M_{ML}$  denotes the median of the set *Med-Low*,  $M_H$  the median of the set *High*, and so on. The medians of the *Output* sets are determined in EEF-CAS Step 2. Fig. 3 shows the ANFIS system into which the rules would be embedded.

**Layer 1:** Each neuron function is a bell membership function that corresponds to the fuzzy set of one of the input attributes, as determined in EEF-CAS Step 2. For example, the first neuron in the first layer is associated with the Attribute 1 fuzzy set  $Low_1$ . Its output is the membership grade of Attribute 1 in its fuzzy set  $Low_1$ .

**Layer 2:** Each neuron in this layer corresponds to the antecedent of one of the rules. For instance, the first neuron corresponds to the antecedent of the first rule,  $A_{R1}$ . Its inputs are the membership grade of Attribute 1 in its fuzzy set  $Low_1$  and the membership grade of Attribute 2 in its fuzzy set  $Medium_2$ . The output is the firing strength of the first rule.



**Figure 3.** An ANFIS system upon which the EEF-CAS has been implemented

**Layer 4:** Each neuron  $i$  in this layer corresponds to the consequent of the rule represented by neuron  $i$  in the second layer. The output is the product of the consequent of the rule it represents and the rule's firing strength.

**Layer 5:** The outputs of the fourth layer neurons are summed and output as the effort estimate.

Once the fuzzy sets, medians, and rules have been embedded into the ANFIS, the historical data collected in EEF-CAS Step 1 are used to train the ANFIS. By training the ANFIS with historical data, the bell-shaped fuzzy sets of the input profiling attributes are fine-tuned by having their  $a$ ,  $b$ , and  $c$  parameters, representing the width, side slope, and center location of the bell function, modified according to historical data. Upon completion of training, the ANFIS is ready to be used for software effort estimation.

## 4. Case study

The evaluation of the proposed EEF-CAS was performed on a use case involving the Industrial Partner, a Fortune 500 company with a workforce of over 100,000 people worldwide and annual revenue of over \$30 billion dollars. The data were collected during the development of three different products and two releases of each, for a total of six projects. The development team consisted of an average of six people, and the team was located on two different continents.

Originally, the Industrial Partner estimated to have approximately 2000 historical data points scattered in different spreadsheet files. Therefore, the data had to be centralized before knowing the true value of the number of data points. Once the data was centralized into one common database, it was discovered that only about 1400 data points existed. Subsequently, data points that contained estimates but no actual values (i.e. bad data), and data points of tasks that did not belong to the implementation phase were also filtered out. Non-implementation tasks were filtered out because finding a common set of attributes for tasks of all phases of the software development cycle would be difficult.

Furthermore, all implementation tasks with estimated or actual effort size of over 100 hours or magnitude of relative error (MRE) greater than 50% were filtered out, leaving only 313 data points. The reason for filtering out tasks with a MRE greater than 50 was that allowing a large range of MRE values required a very high volume of data points to train the neural network. Due to the fact that a high volume of data was not available, the MRE range was limited. Finally, tasks with estimated or actual effort size over 100 hours were filtered out because they were quite rare, and therefore the few data points that did exist would bias the neural network into creating an input-output relationship that was incorrect.

The following sections describe and discuss the four steps of the EEF-CAS application together with the results achieved in each step.

### 4.1. Step 1: Attribute selection

Through collaboration with the Industrial Partner, it was decided to focus on the implementation phase because finding a common set of attributes for tasks of all phases of the software development cycle would be very challenging. After careful analysis of hundreds of metrics used in existing estimation models or researched internally by the Industrial Partner, the profiling attributes were selected for their usefulness, measurability, and significance. Each selected profiling attribute was defined and described using a set of sub-attributes into which it was decomposed. Initially, the following eight attributes were selected: *Skill Level of Implementer*, *Familiarity with Technology*, *Familiarity with Programming Language*, *User Interface*, *Complexity*, *Familiarity with Functionality*, *Familiarity with Domain*, and *Estimated Size*. Examples of profiling attributes (with its sub-attributes) are shown in Tables 6 and 7.

After the profiling attributes were determined, clearly defined, and characterized by the measuring system, they were used to profile historical implementation tasks of the Industrial Partner. During the profiling process, it became apparent that the attribute *Skill Level of Implementer* would be difficult to use because the Industrial Partner considered its evaluation to be a breach of the confidentiality agreement between the employer and the employee. As a result, that attribute was removed from the profiling set.

**Table 6.** Description of Familiarity with Functionality

Name	Familiarity with Functionality, Functionality
Definition	The degree to which the implementer’s familiarity with the functionality influences the estimate.
<b>SUB-ATTRIBUTES</b>	
Similarity	The degree to which the current task resembles something that the implementer has previously implemented.
Product Knowledge	How familiar the implementer is with the application/product being developed. This will provide a measure of how well the implementer understands the effect of this component/functionality on existing components/functionality.
Component Knowledge	How familiar the implementer is with the component involved in the current task.

**Table 7.** Description of Estimated Size

Name	Estimated Size
Definition	The degree to which the size of the task, in hours, influences the estimate.

Because of the low volume of data available to train the neural networks, it was considered beneficial to use as few inputs as possible. Therefore, after all the historical tasks had been profiled, several tests were conducted to see whether any of the profiling attributes could be eliminated. Because the *User Interface* attribute was applicable only to certain implementation tasks and only forty percent of the data made use of it, it was decided that the data were insufficient to evaluate the significance of this attribute correctly. Consequently, the *User Interface* attribute was eliminated.

In addition, several attributes were found to have equal values for a suspiciously large number of data points. It was suspected that the members of the development team who profiled the data perceived some of the “Familiarity with” attributes as very similar and therefore consistently evaluated these attributes similarly. Tests were conducted to evaluate the amount of similarity between the following attributes: *Familiarity with Functionality*, *Familiarity with Technology*, *Familiarity with Domain*, *Familiarity with Programming Language*, and *Complexity*. The *Complexity* attribute was included only to serve as a comparison measure. The results obtained are presented in Table 8.

It is apparent that more than 55% of the data set contains the same values for the attributes *Familiarity with Technology* and *Familiarity with Language*. This degree of similarity between the two attributes was definitely abnormal, and therefore one of them had to be eliminated. It was determined that the attribute which showed the highest similarity with other attributes would be eliminated. At 34.7%, the similarity between the *Technology* and *Domain* attributes was higher than that between the *Language* and *Domain* attributes (31.5%). Furthermore, at 22.3%, the similarity between the *Technology* and *Functionality* attributes was also higher than that between the *Language* and *Functionality* attributes (16.9%). Therefore, the *Technology* attribute was more often perceived to be the same as the *Domain* and *Functionality* attributes, compared to the *Language* attribute. As a result, the *Familiarity with Technology* attribute was eliminated, as shown by the shaded row in Table 7. Similar observations led to removal of the *Familiarity with Domain* attribute.

The remaining attributes of *Functionality* and *Language* had a low degree of similarity between them (16.9%) and were therefore retained. Therefore, the *Familiarity with Functionality*, *Familiarity with Language*, *Estimated Size*, and *Complexity* attributes were used as inputs to neural network training.

**Table 8.** Similarities between profiling attributes

	Familiarity with Language	Familiarity with Domain	Familiarity with Functionality	Complexity
Familiarity with Technology	55.7%	34.7%	22.3%	10.8%
Familiarity with Language		31.5%	16.9%	10.8%
Familiarity with Domain			25.8%	15%
Familiarity with Functionality				9%

## 4.2. Step 2: Data set separation

Each data point was defined by four profiling attributes: *Complexity*, *Functionality*, *Language*, *Estimated Size*, and the output attribute, *Actual Effort*. The qualitative profiling attributes *Functionality*, *Language*, and *Complexity* could take on any value between one and five, or when normalized, between 0.2 and one. The *Estimated Size* and *Actual Effort* attributes could take on any positive value between zero and one hundred. All the attribute values were normalized to be between 0 and 1.

### 4.2.1. Qualitative profiling attributes

The procedure described in Section 3.2 was used to separate the possible range of values into three fuzzy sets: *Low*, *Medium*, and *High*. The boundaries of the Boolean sets were located at the crossover points of each fuzzy set. Table 9 shows the Boolean transformation of the qualitative profiling attributes of the sample data to Boolean sets.

**Table 9.** Sample qualitative attributes transformed into Boolean sets

Complexity		Functionality		Language	
Continuous	Boolean	Continuous	Boolean	Continuous	Boolean
0.4	[1 0 0]	0.2	[1 0 0]	0.2	[1 0 0]
0.9	[0 0 1]	0.4	[1 0 0]	0.8	[0 0 1]
0.2	[1 0 0]	0.866667	[0 0 1]	0.7	[0 1 0]

### 4.2.2. Quantitative attributes

The remaining profiling attribute, *Estimated Size*, and the output attribute, *Actual Effort*, were not separated into sets because they were quantitative in nature. To determine Boolean sets and then fuzzy sets for these attributes, the equal-data-amount criterion was used.

For the *Estimated Size* profiling attribute, the sets into which the data were separated are shown in Table 10. To illustrate the equal data principle that determined the boundaries of the sets, the percentage of data points within each set is also included. The same approach was applied to the output attribute, *Actual Effort*.

**Table 10.** The *Estimated Size* Boolean sets

Set Name	Range of Values in the Set (hours)	% of Data Points	Median	Boolean vector
Low	0 < Estimated Size < 4	18.2%	1.83	[1 0 0 0 0]
Medium-Low	4 <= Estimated Size < 8	17.9%	4.83	[0 1 0 0 0]
Medium	8 <= Estimated Size < 11	17.9%	8	[0 0 1 0 0]
Medium-High	11 <= Estimated Size < 17	16.0%	16	[0 0 0 1 0]
High	17 <= Estimated Size < 30	15.0%	24	[0 0 0 0 1]
Very High	Estimated Size >= 30	16.6%	42	[0 0 0 0 1]

Once the Boolean sets had been determined, the fuzzy sets could be determined for the profiling attribute *Estimated Size*. For each set in Table 9, the median shown was used as the center of the generalized bell membership function, and the width of the membership function was based on the width of the Boolean set. The slope of the sides of each generalized bell function, which is controlled by the parameter  $b$  as described in Section 3.2.1, was set to two for all the sets. During the ANFIS training stage in Step 4, this parameter would be varied to optimize the results.

## 4.3. Step 3: Neural network training

The third step consists of training neural networks with the Boolean data and selecting the network that achieved the most accurate classification results.

To determine the network architecture and the combination of inputs that would most accurately describe the factors affecting software implementation task effort, three parameters were varied during network training: inputs, number of hidden neurons, and number of training epochs.

In an attempt to discover the combination of one or more profiling attributes that best predicted the output of the neural network, the number and combinations of profiling attributes used as input to the neural network were varied. The four profiling attributes used as inputs created fifteen different combinations of from one to four attributes. Therefore, for every other parameter varied during the training phase, fifteen different input combinations had to be tested, resulting in fifteen experiments for each test case. In the figures, the inputs have been abbreviated using the first letter of the attribute name: L for *Language*, C for *Complexity*, F for *Functionality* and E for *Estimated Size*.

To determine the number of hidden neurons, three different neural network architectures were used: one with fifteen neurons in the hidden layer, another with thirty-five, and a final one with fifty neurons.

The final parameter varied was the number of epochs used to train the neural networks. In Test Case 1, the number of epochs was limited to 1000; Test Case 2 involved the same experiments, but the number of epochs was increased to 5000.

Network accuracy was evaluated by means of two measures: classification correctness and the Mean Squared Error (MSE). A correct classification was considered to be a point that was classified into the right output set. Nevertheless, software effort estimation research commonly uses Magnitude of Relative Error (MRE), Mean MRE (MMER), and prediction  $Pred(p)$  [6] as accuracy measures. However, this research used MSE because the Industrial Partner considered it the most suitable, primarily due to familiarity and use in other domains.

#### 4.3.1. Test case 1

Fig. 4 and Fig. 5 show the results of testing all fifteen input combinations and all three network architectures when each of the networks was trained with 1000 epochs. The average percentage of correctly classified data points is shown in Fig. 4 and the average mean squared error (MSE) in Fig. 5. Overall, the trained networks performed poorly: the average percentage of correctly classified data points varied from 5.29% to 33.42%. The networks with only *Estimated Size* as input performed significantly better than the rest of the input combinations, especially with the network architecture with 50 hidden neurons. However, even at the highest data classification success rate of 33.42%, the results were too poor to enable accurate rule extraction from the network.

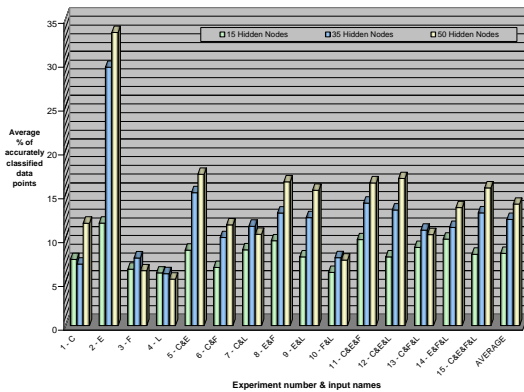


Figure 4. Test Case 1: Classification accuracy

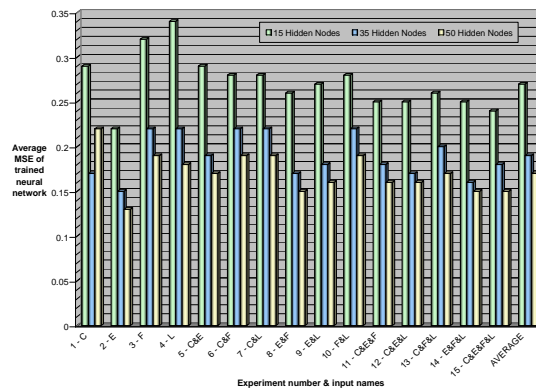


Figure 5. Test Case 1: Mean squared error

In terms of network architecture, Fig. 4 shows that changing the number of neurons in the hidden layer from thirty-five to fifty did not significantly improve network performance. The average increase in classification accuracy was only one percent between the two different network architectures. In fact, in experiments 3, 4, 7, 10, and 13, the network classification accuracy actually decreased with fifty neurons. On the other hand, the network architecture with thirty-five hidden neurons clearly outperformed the network with fifteen hidden neurons, especially in experiments 2, 5, 9, and 12 where the classification accuracy nearly doubled.

The MSE values for this experiment were also high, ranging from approximately 0.12 when only *Estimated Size* was used to 0.35 when *Language* was the only network input.

### 4.3.2. Test case 2

Fig. 6 and Fig. 7 show the results produced by training neural networks of all three architectures and all fifteen input combinations with 5000 epochs.

It is interesting to note that although the average MSE of all experiments decreased significantly, the average percentage of accurately classified data points also decreased for some of the experiments. The classification performance of the network with fifty hidden neurons and *Estimated Size* as the only input increased to 48%, and in turn, all networks that included this profiling attribute as an input showed improved classification performance. For example, in Fig. 6, experiments 2, 5, 8, 9, 11, 12, 14, and 15 all showed higher classification accuracy than the same experiments in Fig. 4, regardless of the network architecture. However, most networks that did not include *Estimated Size* as an input decreased in classification accuracy.

The lowest average MSE value observed was 0.081, when only *Estimated Size* was used as an input to the network architecture with fifty hidden neurons. Once again, this is a significantly high error rate that would not allow accurate rules to be extracted from the network. Further tests were conducted to determine whether increasing the number of training epochs to 10,000 would significantly increase the classification accuracy of any of the experiments; however, at best, only a 3% accuracy increase was observed.

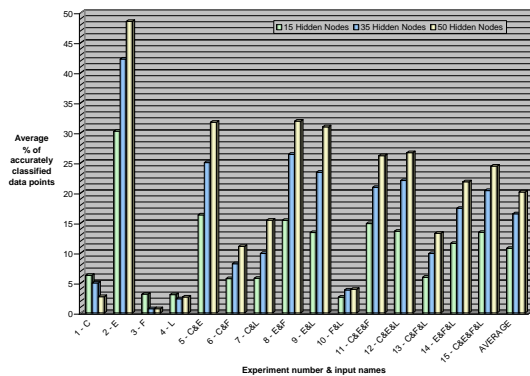


Figure 6. Test Case 2: Classification accuracy

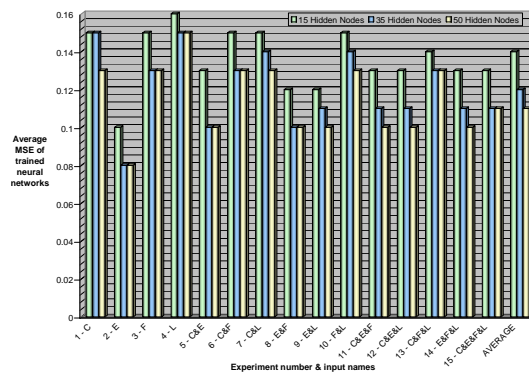


Figure 7 Test Case 2: Mean squared error

### 4.3.3. Analysis and discussion

It was believed that the poor results in Test Cases 1 and 2 could be due to several reasons: poor data set boundaries, lack of data for training the neural networks, or low data quality. Each of these hypotheses was examined, and further tests were conducted when possible.

The first hypothesis stated that the poor results were caused by the poorly located boundaries of the sets into which the attributes *Estimated Size* and *Actual Effort* were separated. Specifically, the hypothesis said that the clustering of the sets of these two attributes, which was based on the equal-data criterion, was inadequate. To test this hypothesis, two more test cases were developed. In these two cases, the *Estimated Size* attribute was not separated into sets, but rather was simply normalized and entered into the neural networks as a value between zero and one. In addition, the output attribute, *Actual Effort*, was also normalized and not separated into sets, meaning that the networks had only one output neuron in their third layers.

This change eliminated any error that was introduced by separating the two quantitative attributes into sets. A data point was considered to be correctly classified if the network output was within 20% of the actual effort. The idea was to emulate a network that estimated implementation tasks that were within 20% of the actual value using what corresponds to the *Pred(20)* metric. This percentage was chosen from the work of Stutzke, which points out that typically an estimate within 20% of the actual effort is adequate for project cost and schedule estimation [20, 39].

Fig. 8 and Fig. 9 show the results of Test Case 3, where 1000 epochs were used to train each network, and Fig. 10 and Fig. 11 shows those of Test Case 4, where 5000 epochs were used. The



average MSE for all the network architectures decreased in both cases compared to the average MSE values of Test Case 1 (Fig. 4 and Fig. 5).

In both Test Cases 3 and 4, the lowest MSE values, 0.0094 and 0.0046 respectively, were achieved when *Estimated Size* was the only input to the network architecture with fifty hidden neurons.

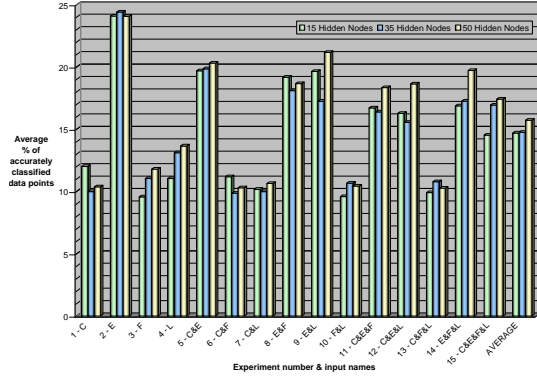


Figure 8. Test Case 3: Classification accuracy

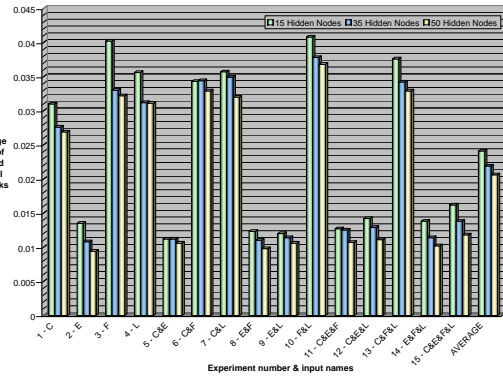


Figure 9. Test Case 3: Mean squared error

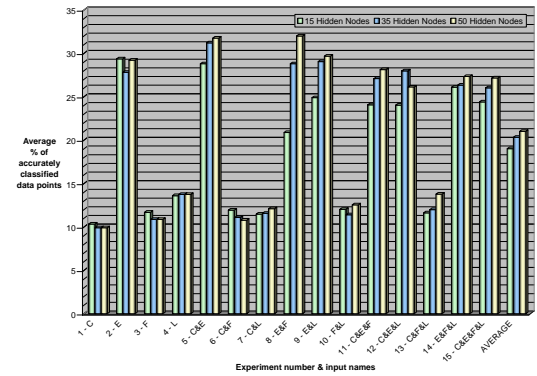


Figure 10. Test Case 4: Classification accuracy

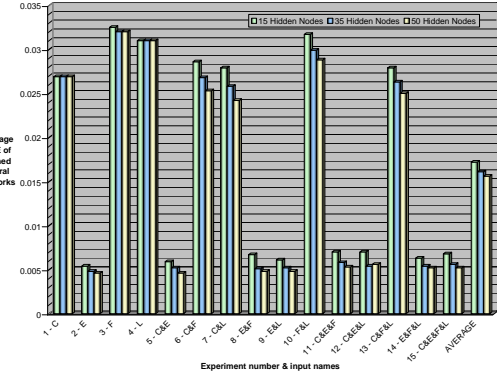


Figure 11. Test Case 4: Mean squared error.

However, despite the lower MSE values, in comparison to Test Cases 1 and 2, the average percentage of correctly classified data points dropped. Overall, while the network was more successfully trained with the data when the quantitative attributes were not separated into sets, the classification accuracy dropped. This shows that the main problem was not the set boundaries determined in the second EEF-CAS step, as assumed in the first hypothesis.

The second hypothesis was that the poor results were caused by the small amount of data. Because no more data points were available, there was no practical way of determining whether more data would yield better results. However, theoretically the findings of Baum and Haussler [25], which were summarized in Section 3.1.2, could be used to test this second hypothesis. Assuming an error of 0.125, for every weighted connection, there should be approximately ten training data points. In the best-case scenario, the network with the smallest number of weighted connections, the network consisted of three neurons in the first layer, corresponding to the sets of the single profiling attribute used as an input, fifteen hidden neurons in the middle layer, and six output neurons in the third layer. This network architecture would total 135 weighted connections, and, as a result, theoretically a minimum of 1350 data points would be required. This is a much larger number than the 313 data points that were available. The third hypothesis, which stated that the quality of the training data might have been low, leading to poor classification results, cannot be proved true or false due to the lack of data. If additional data were collected and if the classification accuracy continued to be low, then the data quality hypothesis could be further investigated.

Overall, the data were insufficient to proceed to the fourth step of the EEF-CAS, rule extraction. However, given that only 313 data points were available, the results are promising.

#### 4.4. Step 4: Rule extraction and ANFIS implementation

EEF-CAS Step 4 could not be implemented in this research due to the low classification accuracy achieved in Step 3. Collection of additional data might improve the network accuracy to a point that would enable rule extraction and ANFIS implementation. However, at the moment, the Industrial Partner is not prepared to invest additional time and effort in data collection. Nevertheless, Section 3.4 describes in detail how the full EEF-CAS implementation would be completed.

#### 5. Limitations

The drawback of the proposed approach is the need to achieve relatively high classification accuracy in the EEF-CAS third step to proceed to the last step, rule extraction and ANFIS implementation. In the present case study, insufficient accuracy prevented completion of the last step. However, this step is optional because the framework is functional without it. An estimate is still produced, although because rule extraction from the neural network has been omitted, and thus cause-effect relationships are not established. Performing this last step would also increase estimation accuracy because the fuzzy sets determined in Step 2 are fine-tuned in this step.

In an attempt to increase practical application of this research as well as to address the lack of real-life studies [1], the present use case consists of a real-life scenario involving a multinational industrial partner. This imposed the use of classification correctness and Mean Squared Error (MSE) as estimation metrics, rather than other measures typically used in software estimation, such as Magnitude of Relative Error (MRE), Mean MRE (MMER), and prediction  $Pred(p)$  [6]. Nevertheless, the industrial environment made it possible to examine the application of the proposed approach in a practical setting.

Moreover, the real-life scenario provided an opportunity to demonstrate how the attribute selection step is conducted. At the same time, this scenario, in conjunction with the organization-specific attribute selection stage, prevented comparison of the estimation accuracy with that of other approaches. An accuracy comparison could possibly have been performed using public data sets such as COCOMO'81 [40] or ISBSG [41]; however, this would have limited how the EEF-CAS attribute selection step could be performed because this stage requires close involvement of the organization implementing the estimation. If a public data set were used, the EEF-CAS attribute selection stage would have been limited to choosing attributes from the available set. Consequently, the definition of attributes and sub-attributes, data collection, attribute analysis, and selection could not have been performed. As a result, an essential part of the EEF-CAS would have been omitted.

#### 6. Conclusions

Most software effort estimation research has focused on improving accuracy [1]. Specifically, much research has focused on formal models [1]; however, there is no conclusive evidence that formal models produce more accurate estimates than expert estimation [3]. Furthermore, expert estimation approaches prevail in industry [1, 2].

Nevertheless, the ability to produce accurate software development effort estimates is essential to the software industry. Based on them project scope is determined, quality standards are set in place, and cost and schedule constraints are defined. Yet, software development effort estimates are often plagued by omissions, uncertainty, and bias [20]. Existing estimation models continue to frequently produce inaccurate estimates, instigating research studies that attempt to determine the properties they lack. After decades of such studies and practical experience, a number of deficiencies have been found that hinder existing estimation models from producing accurate estimates. This research focused on developing a new effort estimation model that amends those deficiencies by incorporating within it the following characteristics [4]:

1. The ability to handle diverse process and product variables.
2. The ability to incorporate empirical evidence and expert judgment.
3. The ability to determine genuine cause and effect relationships.
4. The ability to handle uncertainty.
5. The ability to handle incomplete information.

Furthermore, the proposed Effort Estimation Framework with Customizable Attribute Selection (EEF-CAS) incorporates diverse, company-specific profiling attributes in the attribute selection stage, while empirical evidence is included through neural network training. Fuzzy logic addresses imprecision and uncertainty, and rule extraction from a neural network establishes cause-effect relationships.

The EEF-CAS provides a flexible and customizable framework for software estimation in which components can be adjusted or substituted to accommodate a specific estimation environment. Different system characteristics and personal skills are accommodated in the attribute selection stage, where the estimator selects the factors that it believes most influence the task effort.

Because few estimation studies exist in which evaluations are performed in real-life estimation scenarios [1], the EEF-CAS was evaluated on a case study involving a multinational industrial partner. Even though in the present case study, the last, optional EEF-CAS step could not be performed due to low classification accuracy in the third step, the results are promising. The EEF-CAS framework has satisfied the stated requirements; however, its success is highly dependent on the quality and amount of available data. The use of different, readily available software evaluation data sets might have achieved better classification, but it would have greatly restricted the attribute selection stage and would not have allowed observation of the framework in a real-life setting.

The EEF-CAS as a framework identifies the main steps in the estimation process, providing flexibility of implementation. Consequently, in addition to applying EEF-CAS on a new use case, the authors plan to explore the proposed framework by applying different techniques in some of the steps, for example, exploring self-organizing maps or different clustering algorithms to determine boundaries for the quantitative attributes and making use of algorithms to determine better neural network architectures in Step 3. In conclusion, the Effort Estimation Framework with Customizable Attribute Selection proposed in this paper provides a successful foundation for overcoming many of the obstacles faced by existing software development effort estimation models.

## 7. References

- [1] Magne Jørgensen, Martin Shepperd, "A Systematic Review of Software Development Cost Estimation Studies", *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp.33-53, 2007.
- [2] Kjetil Moløkken, Magne Jørgensen, "A Review of Surveys on Software Effort Estimation", In *Proceedings of the International Symposium on Empirical Software Engineering*, pp.223-230, 2003.
- [3] Magne Jørgensen, "Forecasting of Software Development Work Effort: Evidence on Expert Judgment and Formal Models", *International Journal of Forecasting*, vol. 23, no. 3, pp.449-462, 2007.
- [4] Norman E. Fenton, Martin Neil, "Software Metrics: Roadmap", In *Proceedings of the Conference on the Future of Software Engineering*, pp.357-370, 2000.
- [5] Ekrem Kocaguneli, Ayse Tosun, Ayse Bener, "AI-based Models for Software Effort Estimation", In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp.323-326, 2010.
- [6] Ali Idri, Sanaa Elyassami, "Applying Fuzzy ID3 Decision Tree for Software Effort Estimation", *International Journal of Computer Sciences Issues*, vol. 8, no. 4, pp.131-138, 2011.
- [7] Mohammad Azzeh, Daniel Neagu, Peter I. Cowling, "Fuzzy Grey Relational Analysis for Software Effort Estimation", *Empirical Software Engineering*, vol. 15, no. 1, pp.60-90, 2010.
- [8] Ruchi Shukla, A. K. Misra, "AI-based Framework for Dynamic Modeling of Software Maintenance Effort Estimation", In *Proceedings of the International Conference on Computer and Automation Engineering*, pp.313-317, 2009.
- [9] Parag C. Pendharkar, James A. Rodger, "A Distributed Problem-Solving Framework for Probabilistic Software Effort Estimation", *Expert Systems*, vol. 29, no. 5, pp.492-505, 2011.
- [10] Vishal Sharma, Harsh Kumar Verma, "Optimized Fuzzy Logic-Based Framework for Effort Estimation in Software Development", *International Journal on Computer Sciences Issues*, vol. 7, no. 2, pp.30-38, 2010.
- [11] Moataz A. Ahmed, Moshood Omolade Saliu, Jarallah AlGhamdi, "Adaptive Fuzzy Logic-Based Framework for Software Development Effort Prediction", *Information and Software Technology*, vol. 47, no. 2, pp.31-48, 2005.

- [12] Xishi Huang, Danny Ho, Jing Ren, Luiz F. Capretz, "A Soft Computing Framework for Software Effort Estimation", *Soft Computing*, vol. 10, no. 2, pp.170-177, 2006.
- [13] Gary D. Boetticher, "Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains", In *Proceedings of Model-based Requirements Workshop*, pp.17-24, 2001.
- [14] Gavin Finnie, Gerhard E. Wittig, "AI Tools for Software Development Effort Estimation", In *Proceedings of the International Conference on Software Engineering: Education and Practice*, pp.346-352, 1996.
- [15] Xishi Huang, "A Neuro-Fuzzy Model for Software Cost Estimation", Dissertation, University of Western Ontario, 2003.
- [16] Roheet Bhatnagar, Vandana Bhattacharjee, Mrinal Kanti Ghose, "A Proposed Novel Framework for Early Effort Estimation using Fuzzy Logic Techniques", *Global Journal of Computer Science and Technology*, vol. 10, no. 14, pp.66-71, 2010.
- [17] Magne Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort", *Journal of Systems and Software*, vol. 70, no. 1, pp.37-60, 2004.
- [18] Robert L. Glass, "Facts and Fallacies of Software Engineering", Addison-Wesley, Boston MA, USA, 2003.
- [19] Roger S. Pressman, "Software Engineering: A Practitioner's Approach", McGraw-Hill Higher Education, Boston MA, USA, 2009.
- [20] Richard D. Stutzke, "Estimating Software-Intensive Systems: Projects, Products, and Processes", Addison Wesley, Upper Saddle River NJ, USA, 2005.
- [21] Jessica Keyes, "Software Engineering Handbook", Auerbach Publications, Boca Raton FL, USA, 2003.
- [22] Nesreen Al Khalidi, Ahmad A. Saifan, Izzat M. Alsmadi, "Selecting a Standard Set of Attributes for Cost Estimation of Software Projects", In *Proceedings of the International Conference on Computer, Information and Telecommunication Systems*, pp.1-5, 2012.
- [23] Katrina Maxwell, Luk Van Wassenhove, Soumitra Dutta, "Performance Evaluation of General and Company-Specific Models in Software Development Effort Estimation", *Management Science*, vol. 45, no. 6, pp.787-803, 1999.
- [24] Robert Fullér, "Introduction to Neuro-fuzzy Systems", Physica-Verlag: Heidelberg NY, USA, 2000.
- [25] Eric B. Baum, David Haussler, "What Size Net Gives Valid Generalization", *Neural Computation*, vol. 1, no. 1, pp.151-160, 1989.
- [26] Sivanandam SN, Sumathi S, Deepa SN, "Introduction to Fuzzy Logic using MATLAB", Springer: Berlin and Heidelberg, Germany, 2007.
- [27] Jyh-Shing Roger Jang, Chuen-Tsai Sun, Eiji Mizutani, "Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence". Prentice-Hall, Upper Saddle River NJ, USA, 1997.
- [28] Rand R. Wilcox, "Fundamentals of Modern Statistical Methods: Substantially Improving Power and Accuracy", Springer: New York, USA, 2010.
- [29] Michael Negnevitsky, "Artificial Intelligence: A Guide to Intelligent Systems", 2nd ed. Addison-Wesley, Harlow, UK, 2005.
- [30] Murata N, Yoshizawa S, Amari S, "Network Information Criterion: Determining the Number of Hidden Units for an Artificial Neural-network Model", *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp.865-872, 1994.
- [31] Noboru Murata, Shuji Yoshizawa, Shun-ichi Amari, "Finding the Number of Hidden Neurons for an MLP Neural Network using Coarse to Fine Search Technique", In *Proceedings of the 10th International Conference On Information Sciences, Signal Processing and their Applications*, pp.606-609, 2010.
- [32] Eu Jin Teoh, Cheng Xiang, Kay Chen Tan, "Estimating the number of hidden neurons in a feedforward network using singular value decomposition", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp.1623-1629, 2006.
- [33] Gary D. Boetticher, "An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator", In *Proceedings of the Second Int. Workshop on Soft Computing Applied to Software Engineering*, 2001.

- [34] Robert Andrews, Joachim Diederich, Alan B. Tickle, "Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks", *Knowledge-Based Systems*, vol. 8, no. 6, pp.373-389, 1995.
- [35] Krishnan R, Sivakumar G, Bhattacharya P, "A Search Technique for Rule Extraction from Trained Neural Networks", *Pattern Recognition Letters*, vol. 20, no. 3, pp.273-280, 1999.
- [36] Jyh-Shing Roger Jang, "ANFIS: Adaptive-network-based Fuzzy Inference System", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp.665-685, 1993.
- [37] Michio Sugeno, "Industrial Applications of Fuzzy Control", Elsevier Science Inc., New York, USA, 1985.
- [38] Huajin Tang, Kay Chen Tan, Zhang Yi, "Neural Networks: Computational Models and Applications", Springer: Berlin, Heidelberg, New York, USA, 2007.
- [39] Saleem Basha, Dhavachelvan Ponnurangam, "Analysis of Empirical Software Effort Estimation Models", *International Journal of Computer Science and Information Security*, vol. 7, no. 3, pp.68-77, 2010.
- [40] Barry W. Boehm, "Software Engineering Economics", Prentice-Hall, Englewood Cliffs NJ, USA, 1981.
- [41] International Software Benchmarking Standards Group, ISBSG dataset, <http://www.isbsg.org/>, 2012.