AN AUTOMATED NEURO MODEL FOR SOFTWARE EFFORT ESTIMATION AND RMMI USING COMPETITIVE LEARNING

Mr. M. Senthil Kumar and Dr. B. Chidhambara Rajan

Department of Information Technology, Madurai Institute of Engineering and Technology, Sivaganga
SRM Valliammai Engineering College, Chennai

ABSTRACT: Problem statement: This Paper addresses the neural model for analyzing software effort estimation and risk prediction using the automated neuro tool is proposed. The model carries some of the desirable features of the neural approaches such as learning ability and good interoperability with the use case models of software estimation. It can train the system to calculate the Use case points without use of the automated use case tools and XML input files.

Approach: In this paper, is discussed & compared with the existing work on effort estimation and also it proposes a Neural Network training Approach for effort estimation process with good judgments between use case types.

Results: Knowledge acquired by the network from the environment through a learning process is given as the input.

Conclusion: Furthermore, this method can help provide a path for Risk Mitigation and Monitoring (RMMI) measures and reduction of development cost of the software projects.

Keywords: Effort Estimation, Neural Networks, RMMI, XML files.

INTRODUCTION

Effort Estimation is considered as a promising method for approaching the issue of reliable software cost estimation [18], because it deals with the rapid changes in the software development environments, technologies used, and people involved etc. The Challenging issue for developers and researchers lie in estimating the software effort and Risk Management. Accurate estimates of the cost for software projects are crucial for better project planning, monitoring and control.

The software plays a vital role in the market, since it becomes necessary to develop a package which is used to estimate effort within a short period and with minimum input data. So, the proposed model will be focusing on four modules: 1. Training module 2. Software estimation using Use Case points. 3. Risk Prediction 4. Risk mitigation and management.

Several models and techniques have been proposed and developed [1,4,6]. But the most common methods for software cost estimation are Use Case model and Function Point analysis. Use Case model can be used to predict the size of the future software system, and so Use Case method has been proposed [7,9]. Use Case point method is influenced by the function point’s methods and it is based on analogous Use Case point [5,11].

This paper investigates the existing cost estimation issues and training the systems using Neural Networks to calculate Use Case point from the use Case model. The Use Case model mainly consists of two documents. a) System and b) Subsystem documents. Soft computing provides software developers with some promising techniques such as fuzzy logic, artificial intelligence and neural networks for software estimation [15]. In particular, neural networks is powerful in solving real world applications with imprecise and uncertain information and in dealing with semantic knowledge.

The neural network provides a more powerful tool to solve many issues in software engineering including software estimation [16].

RELATED WORK

Until today, several researches [2,3] and case studies have been reported about the use case point and effort estimation base on use case mode. Smith proposed a method to estimate line of code
from use case diagram [10,12] Arnold and Dross suggested that the use case method can be used to estimate the size of the software [14]. Frank reported the automatic tool using XML file for software effort estimation. Software effort estimation can also be done by function point method which is a measure of software size that logically measures the functional terms. In this approach we need detailed information about the software. Such detailed information of software is difficult to collect at the earlier phase of the software development [5, 13]. To estimate the effort in the earlier phase of the development life cycle process, use case point method has been proposed.

NEURO COMPETITIVE LEARNING MODEL

Our novel neuro-competitive learning is used to train the system for software development effort estimation (shown in Fig. 1). The inputs of this model are actors, use case weights, use case transactions and use case operations. In the neuro-competitive learning process, the output neurons of a neural networks compete among themselves to become active (Fired), and only a single output neuron is active at any one time. It is this feature that makes competitive learning highly suited to estimation of effort in software development.

There are three basic elements to a competitive learning rule.

- A Set of neurons is all the same except for some randomly distributed synaptic weights and so, it responds differently to a given set of input patterns.
- A limit is imposed on the “Strength of each neuron.
- It is a mechanism that permits the neurons to compete for the right to respond to a given subset of input. Such as only one output neuron, or only one neuron per group, is active at a time. The neuron that wins the competition is called a winner- takes all neuron.

USE CASE MODE

The paper is going to focus mainly on two parameters system-level use case diagram and flow of events. System-level use case diagram includes one or more use case diagrams showing all the use cases and actors in the system [8]. Figure (2) shows an example of system – level use case diagram for online Ticket reservation systems.

- A Session is started when a passenger logins into the login menu by giving his/her user name and password.
- A system checks the username and password with the database for validation.
- If the username or password is wrong, then an error message is displayed on the screen, and the operation is aborted.
- Otherwise, a passenger is allowed to perform one or more transactions, choosing from a menu of possible type of transaction in each case.

4.1. COUNTING USE CASE POINT

Intuitively, UCP is measured by counting the number of actors and transactions included in the flow of events with some weight. A transaction is an event that occurs between the actor and the target system [17].

MATERIALS & METHODS

Step 1: Counting actors

The actors in the use case are categorized as simple, average or complex. A simple actor represents another system with a defined weight. A complex actor is a person interacting through a GUI interface [17].
The number of each actor type that the target software includes is calculated and then each number is multiplied by a weighting factor shown in Table 1. Finally the actor’s weight is calculated by the adding those values together. This table is given as one of the inputs to the competitive learning method.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>Program interface</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>Interactive, or protocol driver</td>
<td>2</td>
</tr>
<tr>
<td>Complex</td>
<td>Graphical user interface</td>
<td>3</td>
</tr>
</tbody>
</table>

Step 2: Counting use case weights

Each use case should be categorized into simple, Average or Complex based on the number of transactions including the alternative paths. A simple Use case has 3 or fewer transactions, an average use case has 4-7 transactions and a complex use case has more than 7 transactions. Then the number of each use case type is counted and then each number is multiplied by a weighting factor shown in table 2. This table is also given as one of the inputs to the competitive learning method.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>3 or fewer transactions</td>
<td>5</td>
</tr>
<tr>
<td>Average</td>
<td>4 to 7 transactions</td>
<td>10</td>
</tr>
<tr>
<td>Complex</td>
<td>More than 7 transactions</td>
<td>15</td>
</tr>
</tbody>
</table>

Step 3: USE Case transactions and operations

The use case transactions and use case operations are also fed as input to the competitive learning method, for example, transactions like enquiry, reservation operations like save () and exit (). Our mechanism permits the each neuron to compete for the right and to respond to a given subset of inputs, such that only one neuron, or only one neuron per group (from each input group) is active at a time.

Step 4: Calculating Unadjusted Use Case Points

It is calculated by adding the total weights for actors to the total for use case weights use case transactions & use case operations (Figure 1)

Step 5: Weighting Technical and environmental Factors

The UUCP are adjusted based on the values assigned to the number of technical and environmental factors shown in table 3 & 4.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Distributed system</td>
<td>3</td>
</tr>
<tr>
<td>T2</td>
<td>Response or throughput performance objectives</td>
<td>4</td>
</tr>
<tr>
<td>T3</td>
<td>End-user efficiency (online)</td>
<td>5</td>
</tr>
<tr>
<td>T4</td>
<td>Complex internal processing</td>
<td>2</td>
</tr>
<tr>
<td>T5</td>
<td>Code must be readable</td>
<td>3</td>
</tr>
<tr>
<td>T6</td>
<td>Easy to install</td>
<td>5</td>
</tr>
<tr>
<td>T7</td>
<td>Easy to use</td>
<td>5</td>
</tr>
<tr>
<td>T8</td>
<td>Portable</td>
<td>2</td>
</tr>
<tr>
<td>T9</td>
<td>Easy to change</td>
<td>5</td>
</tr>
<tr>
<td>T10</td>
<td>Concurrent</td>
<td>1</td>
</tr>
<tr>
<td>T11</td>
<td>Includes special security features</td>
<td>4</td>
</tr>
<tr>
<td>T12</td>
<td>Provides direct access for third parties</td>
<td>2</td>
</tr>
<tr>
<td>T13</td>
<td>User training facilities required</td>
<td>2</td>
</tr>
</tbody>
</table>

Step 6: Calculation of TCF

It is calculated by multiplying the value of each factor (t1, t3) in table 3 by its weight and then adding all these numbers to get the sum called the T factor. Finally the following formula is applied

\[
TCF=0.6+ (0.01 * T \text{ factor})
\]  

Step 7: Calculation of environmental factor

It is calculated by multiplying the value of each factor (f1-f8) in table 4 by its weights and adding all the products to get the sum called the E factor. Finally the following formula is applied
Step 8: Estimating Effort

By multiplying the specific value (man-hours) by the UCP, the effort can be easily calculated,

\[ \text{UCP} = \text{UUCP} \times \text{TCF} \times \text{EF} \]  

(3)

PROPOSED METHOD

Based on this method, the scholars have planned to train the system with different input values (Use Case Models) to the learning system. The inputs are actors, use case weights, transaction and operations and the output of the learning competitive method is unadjusted use case points, which are fed as input to effort estimation for software.

In the learning process, the system must learn to differentiate and judge between the type of actors like simple, average and complex. Based on the value of the actors, the unadjusted use case points can be calculated. The technical properties of the project may differ from every project. So, depending on the project, the technical properties of the system is calculated. Both unadjusted use case point value and technical complexity factor gives the size of the project. The number of workers focuses on the team properties. Finally, the effort of the project is decided by the size of the project and the number of workers.

AUTOMATED COMPARISON TOOL USING NEURAL NETWORKS:

There were several existing tools available for Use case point calculation; the scholars are planning to create an automated performance comparison tool using neural network concepts to make the judgment about the use case point analysis. Also it is necessary to train the system by giving the different use case models. The learning algorithm must be effective in judging the type and assigning the factors.

The output of the neural learning process is processed for complexity and it produces the unadjusted use case points, and then adjusted use case points are calculated by using the technical and environmental factors. The result is compared with the performance of the existing available methods.
RESULTS AND LIMITATIONS

In order to evaluate the usefulness of the neuro based effort estimation, the scholars have collected use case models from live software projects to train the system for calculating unadjusted use case points in shorter time with high judgements. In this evaluation, they mainly point out the result of neuro system to classify the types of actors and use cases. Then they have compared the result value of our tool with ones calculated by an existing specialized use case counting. The main limitation of their system that is takes some time for training the system and it needs many real use case models as inputs to make the system as a best classifier between them.

CONCLUSION AND FUTURE WORK

This work attempts to propose the use of neural networks in the estimation of software effort in the earlier phase of development and to evaluate the performance of the method with the existing models. Future work will utilize these results for managing the risk and reduce the cost for the development of the project which is more needed work for the current software development projects.

REFERENCES


