SOFTWARE EFFORT ESTIMATION USING ANALOGY

1JAYASHREE K.M, 2SATHYA BAMA B, 3J. FRANK VIJAY

1,2Department of Information Technology KCG College of Technology Chennai, Tamil Nadu, India
3Professor and Head of IT Department KCG College of Technology Chennai, Tamil Nadu, India
Email: jayashree9212,sathyab93@gmail.com

Abstract—Software Effort Estimation is the process of anticipating the most practical use of effort needed to develop or maintain software. Effort estimation can be done in different ways. Some of them are expert judgment, estimation by analogy, estimation using algorithmic models, etc. In this paper we propose a simple approach for estimating the effort required to develop a software by using analogy.

Keywords- Effort, Analogy.

I. INTRODUCTION

Software Effort Estimation involves forecasting the effort required to develop or maintain a software system. Effort estimation is needed for observing the project’s advancement and evaluating its productivity. Effort estimation methods are classified into algorithmic models and non-algorithmic models. Algorithmic models use mathematical formulae to estimate the effort and software cost. Some of the algorithmic models are COCOMO, Function Point method, Use Case Point method, Lines of Code, etc. But these methods have certain disadvantages such as they need to be adjusted according to the environment and they are not understandable. Non-algorithmic models are developed to overcome these drawbacks. Non-algorithmic models are easier to understand and are capable of exhibiting the association between cost drivers and effort. They also have the ability to extract values from databases. They are generally based on prediction process. Neural networks, stepwise regression and case-based reasoning are some examples.

This paper focuses on case-based reasoning. Case-based reasoning estimates the effort required to develop new software based on comparison with past projects effort values. This concept is termed as estimation by analogy.

The rest of this paper is organized as follows: Section II gives a short introduction on analogy and case-based reasoning, section III gives a detailed explanation about estimation by analogy and the last section IV gives the final conclusion.

II. ANALOGY AND CASE-BASED REASONING

Analogy is the comparison between one subject and another mainly with the intention of description and clarification. In Merriam-Webster’s dictionary the word Analogy is defined as “inference that if two or more things agree with one another in some respects they will probably agree in other”. This states that if two or more things are similar with respect to some characteristics their similarity may also extend to other characteristics. The Analogy-based reasoning is also known as case-based reasoning (CBR). It is the process of deriving the solution by identifying similar cases, and reusing the solution in a new problem.

Analogy-based reasoning or CBR is a four stage cycle. The stages are:
1. RETRIEVE- identifying the cases that are most similar to the target problem.
2. RESUSE- using the past information and solution to solve the new problem.
3. REVISE- adding changes to the the proposed solution so that it adapts better to the target problem.
4. RETAIN- save the parts of current experience in the case-base for future problem solving.

III. ESTIMATION BY ANALOGY

Software effort estimation using analogy is an approach of estimating the effort required to develop a software using the effort values of previous projects that are similar to the new one. It based on the principle “similar projects require similar effort”. It involves four steps. They are explained below.

IV. INPUT GATHERING

- Characterize all projects in the historical database by a set of attributes namely Team Experience, No: of developers, Web Pages, No: of images and No: of animations in the software to be developed. These attributes are relevant, independent, comprehensive and operational.
Software Effort Estimation Using Analogy

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team experience</td>
<td>Number of years experience the team has in web development</td>
</tr>
<tr>
<td>No: of developers</td>
<td>Number of team members</td>
</tr>
<tr>
<td>Web Pages</td>
<td>Number of web pages in the software to be developed</td>
</tr>
<tr>
<td>No: of images</td>
<td>Number of images in the software to be developed</td>
</tr>
<tr>
<td>No: of animations</td>
<td>Number of animations in the software to be developed</td>
</tr>
</tbody>
</table>

- Get the values for these attributes from the user for the candidate project.

Example of values entered by the user:

<table>
<thead>
<tr>
<th>Team experience</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>No: of developers</td>
<td>6</td>
</tr>
<tr>
<td>Web Pages</td>
<td>6</td>
</tr>
<tr>
<td>No: of images</td>
<td>3</td>
</tr>
<tr>
<td>No: of animations</td>
<td>2</td>
</tr>
</tbody>
</table>

- Convert these attribute values into normalized values, i.e., values ranging from 0 to 1 for the purpose of data reduction.
- Normalization is done using the formula given below:

\[ N(v) = \frac{v - \text{min}}{\text{max} - \text{min}} \]

where \( v \) denotes the original value of numerical attribute, and \( \text{min} \) and \( \text{max} \) are the minimum and maximum values of the domain of the attribute.

Values entered by the user after normalization:

<table>
<thead>
<tr>
<th>Team exp</th>
<th>No: of dev</th>
<th>Web Pages</th>
<th>No: of img</th>
<th>No: of ann</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.26</td>
<td>0.8</td>
<td>0.42</td>
<td>0.5</td>
<td>0.33</td>
</tr>
</tbody>
</table>

- Identifying similar cases:

- The value of the first attribute entered by the user is selected.
- It is then compared with the value of the corresponding attribute in the past project that is available in the database.
- If the difference between those values is within a fixed limit, the value of the next attribute is compared and the process proceeds likewise.
- Else, the attribute values are compared with that of the next past project.

Example of database containing past project data:

<table>
<thead>
<tr>
<th>Id</th>
<th>Team exp</th>
<th>No: of dev</th>
<th>Web Pages</th>
<th>No: of img</th>
<th>No: of ann</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.28</td>
<td>0.20</td>
<td>0.14</td>
<td>0.25</td>
<td>0.33</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>0.57</td>
<td>0.80</td>
<td>0.28</td>
<td>0.50</td>
<td>0.33</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>0.14</td>
<td>0.80</td>
<td>0.71</td>
<td>0.75</td>
<td>0.66</td>
<td>17</td>
</tr>
</tbody>
</table>

- Finding the distance:

- If the differences in values of all the attributes are within the fixed range (±0.4), that past project is said to be similar to the new candidate project.
- Identifying similar projects for our example:
  - Project 1 is not similar to the candidate project as the difference in values of the second attribute crosses 0.4.
  - Project 2 and 3 are similar to the candidate project as the difference in values of all the attributes lie within the limit of ±0.4.
- There can be more than one similar past project for each new project. All the similar projects are identified.
- For each similar project, its distance with the new candidate project is calculated by summing up the differences in individual attribute values.
- The distance is found using the formula given below:

\[ d(P,P_i) = \sum_{k=1}^{n} \text{diff}(A_k, A_{ik}) \]

where \( d(P,P_i) \) denotes the distance between the candidate project \( P \) and any past project \( P_i \). \( A_k \) represents the value of \( k \)th attribute in candidate project and \( A_{ik} \) represents the value of \( k \)th attribute in the past project \( P_i \).

Finding the distance with similar projects in our example:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team experience</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>No: of developers</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Web Pages</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>No: of images</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>No: of animations</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Calculating overall distance of candidate project with:

- Project 2:
  \[ 0.29 + 0 + 0.14 + 0 + 0 = 0.43 \]
  Therefore \( d(P,P_2) = 0.43 \)

- Project 3:
  \[ 0.14 + 0 + 0.29 + 0.25 + 0.33 = 1.01 \]
  Therefore \( d(P,P_3) = 1.01 \)

### VII. CALCULATING THE EFFORT

The effort value of the new candidate project is predicted from the effort values of those past projects that were found to be similar to the new one and from the calculated distance values using the following formula.

\[
\text{Effort}(P) = \frac{\sum d(P, P_i) \times \text{Effort}(P_i)}{\sum d(P, P_i)}
\]

Thus for our example the effort is calculated as follows:

\[
\text{Effort}(P) = \frac{(0.43 \times 14) + (1.01 \times 17)}{0.43 + 1.01} = \frac{23.19}{1.44} = 16.10
\]

- The value obtained is then rounded off to nearest whole number.
- Therefore the effort required to develop the new software specified by the user is estimated as 16 man hours.

### CONCLUSIONS

This proposed approach is based on reasoning by analogy. This process is very simple and also less time consuming since it estimates the new effort value only based on comparison with the existing data. It can also produce more accurate results.

### REFERENCES


★★★

Software Effort Estimation Using Analogy

25