Integrating Function Point Project Information for Improving the Accuracy of Effort Estimation

Faheem Ahmed, Salah Bouktif, Adel Serhani, Issa Khalil
College of Information Technology, UAE University, AL Ain, United Arab Emirates
{f.ahmed, salahb, serhanim, ikhalil}@uaeu.ac.ae

Abstract

Software organizations are putting efforts to improve the accuracy of the project cost estimation. This in turn helps them to allocate resources. Software cost estimation has been an area of key interest in software engineering community. Many estimation models divided among various categories have been proposed over a period of time. Function Point (FP) is one of the useful software cost estimation methodology that was first proposed twenty-five years ago using the project repository that contained information about various aspects of software project. In the last twenty five years software development productivity has grown rapidly but the complexity weight metrics values assigned to count standard FP still remain same. This fact raises critical questions about the validity of the complexity weight values and accuracy of the estimation process. The objective of this work is to present a genetic algorithm based approach to calibrate the complexity weight metrics of FP using the project repository of International Software Benchmarking Standards Group (ISBSG) dataset. The contribution of this work shows that information reuse and integration of past project’s function-point structural elements improves the accuracy of software estimation process.

1. Introduction

There are many factors that have an impact on the success of software projects. One of those factors is effort estimation in early development phase. Accurate software estimation is critical for project success because allocation of resources is heavily based on this activity. On the other hand an inaccurate estimation is not only the potential threat to the project failure but also has the potential to thwart the business of even a small software organization. Software engineering research community has presented many models for estimating software development effort and some of them are arguably the most popular such as COCOMO [1], SLIM [2], and FP [3]. These models fall into the category of algorithmic models. This is because specified formulas for estimating development efforts are calibrated using historical data gathered from software industry. FP is an ideal software size metric to estimate cost because it can be used in the early development phase, such as requirements engineering, system analysis, and most importantly it is programming language independent [4].

The complexity weight metrics values of FP reflect the functional size of software. They have been introduced in 1979 based on the historical data of software projects, and have been applied universally. Over the passage of time software development methods have evolved steadily. The methodologies to develop software now a day is differ drastically from what it was over two decades ago. For example the object-oriented paradigm has incorporated into mainstream of software development. Visual programming languages have reduced the complexity of designing graphical user interface applications. Database applications are user friendly because complex reports are easily generated using automatic report generation tools. CASE tools have further reduced the efforts spent in the software development life cycle. In addition to the fact that the complexity weight metrics of FP was set twenty five years ago, the fundamental changes in software development further supports the requirements of calibrating complexity weight metrics of FP. Such a calibration will be more rigorous and accurate by using the software project data from the recent past. The underlying hypothesis foresees a more accurate and reliable estimation process.
In this work we experimentally validate that integrating the information of FP project repository in calibration FP complexity weight metrics values improve the cost estimation of software projects by reducing the difference between estimated and actual cost. The paper is organized as follows. In Section 2 we present the related work. In Section 3 we give a short description of the FP calibration problem. In Section 4 we describe a new genetic algorithm based approach to calibrate the FP estimation model. Experimental setting and results are given in Section 5. Conclusion is drawn in Section 6.

2. Related Work

Software estimation models combining algorithmic models with machine learning approaches, such as neural networks and fuzzy logic, have been viewed with scepticism by the majority of software managers [5]. Srinivasan and Fisher [6] illustrate a neural network learning approach to estimate software development effort known as back-propagation. They indicate possible advantages of the approach relative to traditional models, but also point out limitations that motivate continued research. Furthermore, MacDonell [7] also considers the applicability of fuzzy logic modelling methods to the task of software source code sizing, and suggests that fuzzy predictive models can outperform their traditional regression-based counterparts, particularly with refinement using data and knowledge. The theory of fuzzy sets [8] as been successfully applied to other models of software cost estimation, such as f-COCOMO [9] and NF-COCOMO [10], is sufficiently general to be extended to the well-known FP method. Finnie et al. [11] reported research on the combination of machine learning approach with FP; they compared three estimation techniques using FP as an estimate of system size. Yau and Tsai [12] introduce a fuzzified FP analysis model to help software size estimators to express their judgment and use fuzzy Bspline membership function to derive their assessment values. The weak point of this work is that the others used limited in-house software to validate their model, which brings a great limitation regarding the validation of their model. Lima et al. [13] also propose the use of concepts and properties from fuzzy set theory to extend FP analysis into a fuzzy FP analysis. Al-Hajri et al. [14] establish a new FP weight system using artificial neural network.

3. Function Points: A Short Description

Counting FP requires the identification of five types of functional components: Internal Logical Files (ILF), External Interface Files (EIF), External Inputs (EI), External Outputs (EO) and External Inquiries (EQ). Each functional component is classified as a certain complexity based on its associated file numbers such as Data Element Types (DET), File Types Referenced (FTR) and Record Element Types (RET). The complexity matrix for the five components is shown in Table 1. Table 1 illustrates how each function component is then assigned a weight according to its complexity. The Unadjusted Function Point (UFP) is calculated with Equation 1, where $W_{ij}$ are the complexity weights and $Z_{ij}$ are the counts for each function component.

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} \cdot W_{ij}$$ \hspace{1cm} (1)

Once calculated, UFP is multiplied by a Value Adjustment Factor (VAF), which takes into account the supposed contribution of technical and quality requirements. The VAF is calculated from 14 General System Characteristics (GSC), using Equation 2; The GSC includes the characteristics used to evaluate the overall complexity of the software.

$$VAF = 0.65 + 0.01 \sum_{i=1}^{14} C_i$$ \hspace{1cm} (2)

$C_i$ is the Degree of Influence (DI) rating of each GSC.

Finally, a FP is calculated by the multiplication of UFP and VAF, as expressed in Equation 3.

$$FP = UFP \times VAF$$ \hspace{1cm} (3)

In order to calibrate the FP model, we are concerned with the following optimization problem:

$$\text{Minimize } f(x_1, x_2, \ldots, x_m)$$ \hspace{1cm} (4)

where $f$ is a performance criterion that estimates the accuracy of the FP model, e.g., the Mean Magnitude Relative Error (MMRE), and $x_i, i=1..m \ (m=15)$, are real numbers representing the function point complexity weights. Finding optimal weights that minimize the MMRE performance criterion is known as
assignment problem which is NP-hard. Genetic algorithms have been fairly successful at solving such a problem.

4. Genetic Algorithm Based Approach

Genetic Algorithms are based on the Darwinian theory of evolution whereby species compete to survive and the fittest get a higher chance to remain until the end and produce progeny. Typically, the basic idea of a GA is to start from a set of initial solutions, and to use biologically inspired evolution mechanisms to derive new and possibly better solutions [15]. The derivation starts by an initial solution set \( P_0 \) (called the initial population), and generates a sequence of populations \( P_1, \ldots, P_T \), each obtained by “mutating” the previous one. Elements of the solution sets are called chromosomes. The fitness of each chromosome is measured by an objective function called the fitness function. Each chromosome (possible solution) consists of a set of genes.

**GENETIC_ALGORITHM**

1. Initialize \( P_0 \)
2. BESTFIT \( \leftarrow \) fittest chromosome of \( P_0 \)
3. BESTFITEREVER \( \leftarrow \) BESTFIT
4. for \( t \leftarrow 0 \) to \( T \)
   5. \( Q \leftarrow \) pairs of the fittest members of \( P_t \)
   6. \( Q' \leftarrow \) offspring of pairs in \( Q \) using crossover and mutation
   7. replace the weakest members of \( P_t \) by \( Q' \) to create \( P_{t+1} \)
8. BESTFIT \( \leftarrow \) fittest chromosome in \( P_{t+1} \)
9. if BESTFIT is fitter than BESTFITEREVER then
10. BESTFITEREVER \( \leftarrow \) BESTFIT
11. return BESTFITEREVER

Figure 1: Summary of a genetic algorithm.

At each generation, the algorithm selects some pairs of chromosomes using a selection method that gives priority to the fittest chromosomes. On each selected pair, the algorithm applies one of two operators, crossover and mutation, with probability \( p_c \) and \( p_m \), respectively, where \( p_c \) and \( p_m \) are input parameters of the algorithm. The crossover operator mixes genes of the two chromosomes, while the mutation operator randomly modifies certain genes. Each selected pair of chromosomes produces a new pair of chromosomes that constitute the next generation. The fittest chromosomes of each generation are automatically added to the next generation. The algorithm stops if a convergence criterion is satisfied or if a fixed number of generations are reached. The algorithm is summarized in Figure 1.

To apply a GA to a specific problem, elements of the generic algorithm of Figure 1 must be customized and adapted to the problem. In particular the solutions must be encoded into chromosomes, the two operators (crossover and mutation) and the fitness function must be defined. In this paper, we consider a chromosome as a vector of real parameters where a gene corresponds to a real number representing one FP complexity weight. A chromosome is represented by a vector \( X = (x_1, \ldots, x_m) \), in our case \( m = 15 \).

4.1. Crossover Operator

Crossover is a reproduction operator that happens with high probability \( p_c \). It takes two parent chromosomes and produces one or two child chromosomes. One of the strength of real-coded GAs over the binary coded GAs is the greater freedom to use different techniques of performing crossover and mutation. Based on the chromosome representation, we define for our problem three types of crossover techniques; two-cut point crossover, uniform crossover and arithmetic crossover. Two-cut point crossover is standard way to perform the crossover between the chromosomes. It consists in cutting at the same positions each of the two parent chromosomes into two subsets of genes (vector of real numbers in our case). Two new chromosomes are created by interleaving the subsets. In the uniform crossover two parent chromosomes give birth to a single offspring. Each gene of the new offspring is a copy of the gene at the same position of one parent select randomly. In the arithmetic crossover the genes of offspring chromosomes are obtained by linear combinations of the genes of the two parent chromosomes.

4.2. Mutation Operator

Mutation is the second reproduction operator that happens with a small probability \( p_m \). It is employed to extend the search space by creating new points that could be potential solutions. When a chromosome is chosen for mutation, a random choice is made of some of the genes of the chromosome, and these genes are modified. In our problem, the mutation operator randomly selects certain positions in the real parameters vector representing the chromosome being
mutated. A chosen parameter $x_i$ is mutated by making a positive or negative variation of its value; the direction of this variation is determined randomly with a probability of 0.5. Its amplitude is chosen using a uniform distribution over the interval $[x_i, M.x_i]$, where $M$ is a parameter of our algorithm indicating the maximum variation of a mutated parameter. $M$ is called size of mutation.

4.3. Initial Population and Selection Method

Before starting its evolution process, the GA needs to build an initial population $P_0$ of solutions (vectors of real parameters). In our problem, a member of the initial population $P_0$ is obtained by “perturbating” the original vector $V_0$ of FP complexity weight values shown in Table 1 as:

$V_0=<3.0;4.0;6.0;4.0;5.0;7.0;3.0;4.0;6.0;7.0;10.0;15.0;5.0;7.0;10.0>$

Table 1: Function Point Complexity Weight Metrics

<table>
<thead>
<tr>
<th>Component</th>
<th>Low</th>
<th>Ave</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Inputs</td>
<td>3.0</td>
<td>4.0</td>
<td>6.0</td>
</tr>
<tr>
<td>External Outputs</td>
<td>4.0</td>
<td>5.0</td>
<td>7.0</td>
</tr>
<tr>
<td>External Inquiries</td>
<td>3.0</td>
<td>4.0</td>
<td>6.0</td>
</tr>
<tr>
<td>Internal Logical Files</td>
<td>7.0</td>
<td>10.0</td>
<td>15.0</td>
</tr>
<tr>
<td>External Interface Files</td>
<td>5.0</td>
<td>7.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

For each coordinate of the real parameter vector, we simply use a random number generator to obtain a new value between 0 and 10 times the original coordinate value in $V_0$. The size of the initial population is a parameter of our algorithm, set to 50 after several tuning runs. The process of evolution starts by selecting pair of chromosomes according to the roulette-wheel selection technique, whereby one can imagine a roulette wheel where all chromosomes are placed. Each chromosome is assigned a portion of the wheel that is proportional to its fitness. A marble is thrown and the chromosome where the marble halts is selected.

4.4. Fitness Function

In order to measure the performance of a chromosome representing a parameter vector, the Mean Magnitude of Relative Error (MMRE) is chosen to be our fitness function. It is defined in this paper as follows:

1. Relative Error (RE): For project $i$, Relative Error (RE) measures the estimation deviation and is defined as follows:

$$RE_i = \frac{Estimated_i - Actual_i}{Actual_i}$$

2. Magnitude of Relative Error (MRE): For project $i$, the Magnitude of Relative Error (MRE) measures the absolute estimation accuracy and is defined as follows:

$$MRE_i = \left|\frac{Estimated_i - Actual_i}{Actual_i}\right|$$

3. Mean Magnitude of Relative Error (MMRE): For $n$ projects, the Mean Magnitude of Relative Error (MMRE) is expressed as follows:

$$MMRE = \frac{1}{n} \sum_{i=1}^{n} \left|\frac{Estimated_i - Actual_i}{Actual_i}\right| = \frac{1}{n} \sum_{i=1}^{n} MRE_i$$

5. Experimental Settings & Results

5.1. ISBSG Dataset (Data Population)

In order to compute MMRE we use, in this work, a subset from ISBSG dataset release 9 repository containing function-point data for 600 projects. The subset quality rating is “A” or “B”, the counting method is International Function Point User Group (IFPUG), the effort resource level is one, and the development type is new development or re-development. The quality rating “A” and “B” narrates that there is either no or some minor issues related to integrity of the data. The resource level of one indicates that data represents the development team efforts including project team, project management and project administration team. An effort estimation equation is extracted based on the data subset using statistical regression analysis. The statistical regression analysis assumes that the underlying data are normally distributed.

5.2. Algorithmic Setting and Results

After several tuning runs a number of GA parameters are set in order to achieve better results. To form successive generations, the elitist strategy was used. In iteration, the entire population is replaced, except for a small number $Ne$ of the fittest
chromosomes. The number of generations $T$ was set to 500. The maximum number of chromosomes in a generation was set to 150 by deleting the worst individuals in order to have a reasonable execution time. For the sake of better results the used crossover operations are 40% two-cut point, 20% uniform and 20% arithmetic, and the parameters values of $N_c$, $p_c$ (crossover probability), $pm$ (mutation probability) and $M$ (mutation size) were set to 10, 0.7, 0.2 and 2 respectively.

To accurately estimate the accuracy of generated solutions (real parameter vectors), we used 10-fold cross validation. In this technique, the data set is randomly split into 10 subsets of equal size (60 points in our case). A parameter vector $VT$ is derived by using the union of 9 subsets (acting as training dataset), and tested on the remaining subset. The process is repeated for all the 10 possible combinations and mean and standard deviation values are computed for the $MMRE$ of $V0$ and $VT$ on both the training and the test sample. Table 2 illustrates the results of the experiments using 10-fold cross validation to estimate the mean magnitude of the relative error.

### 6. Conclusion

Software engineering community has been showing keen interest in developing methodologies to improve the cost estimation process of the software project because of its profound impact on the management aspects. The tremendous growth in software development has allowed maintaining large repositories of project related information such as ISBSG dataset which stores information about software size. In this work we experimentally validate that using the information of the FP data repository we can further improve the software cost estimation process by calibrating the complexity weight metrics of FP. Our experiments show an average improvement of 50% in mean magnitude of relative error in cost estimation of software projects using the genetic algorithm approach.

### Table 2: Results of the Experiment

<table>
<thead>
<tr>
<th>Folds</th>
<th>MMRE(V0)</th>
<th>MMRE(VT)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training</td>
<td>Test</td>
</tr>
<tr>
<td>1</td>
<td>91.56%</td>
<td>92.20%</td>
</tr>
<tr>
<td>2</td>
<td>91.68%</td>
<td>91.08%</td>
</tr>
<tr>
<td>3</td>
<td>91.63%</td>
<td>91.60%</td>
</tr>
<tr>
<td>4</td>
<td>91.68%</td>
<td>91.14%</td>
</tr>
<tr>
<td>5</td>
<td>91.56%</td>
<td>92.25%</td>
</tr>
<tr>
<td>6</td>
<td>91.55%</td>
<td>92.33%</td>
</tr>
<tr>
<td>7</td>
<td>91.65%</td>
<td>91.39%</td>
</tr>
<tr>
<td>8</td>
<td>91.67%</td>
<td>91.22%</td>
</tr>
<tr>
<td>9</td>
<td>91.69%</td>
<td>91.07%</td>
</tr>
<tr>
<td>10</td>
<td>91.59%</td>
<td>91.96%</td>
</tr>
<tr>
<td>Mean</td>
<td>91.63%</td>
<td>91.62%</td>
</tr>
</tbody>
</table>

### References


