CHAPTER 4

THE FUZZY APPROACH TO FUNCTION POINT ANALYSIS METHOD.

4.1 INTRODUCTION

*Estimating size is the heart of the software project estimating process* [191].

This statement emphasizes the importance of the size estimation of the project, as it is the first step in any software estimation. This must be done accurately and realistically as this forms the basis for further estimation that is what will be the effort required, the cost of the project and the project schedule. The size is the key parameter for estimation of other activities. If size is not estimated properly, it may have a serious impact on other resource estimation activities. No doubt then estimating size is the heart of the software project.

 Ideally size estimation or ‘sizing’ is best done when there is complete information about the system but unfortunately this is not available till the system is actually built [136]. The estimation of size is very critical and difficult area of project planning. It has been recognized as a critical step from the very beginning. The difficulties in establishing units for measuring size lies in the fact that the software is essentially abstract: It is difficult to identify the size of the system. Size estimation done towards the end of the project would be most accurate but most useless as the estimate is too late and cannot be used for any planning or
decision making. The challenge therefore lies in estimating the size of the software at the beginning of the project when there is only partial information.

Accurately predicting software project costs has long exercised industrialists and researchers alike. The software engineering industry has a poor track record for estimating both effort and delivery dates [136]. Never the less project managers and software development organizations have a great need for estimates at a very early stage in a project in order to appropriately tender for business and to properly manage resources.

4.1 TWO VIEWS OF SIZING

4.2.1 LINES OF CODE

This is the simplest measure of size. But this measure is quite debatable as comment lines, data declarations; blank lines may or may not be included in this [70,85]. LOC is also dependent on the language. [186,222,]. Most researchers agree that LOC should not include comments or blank lines, but all the same, we all know that including comments affects the maintenance costs.

Thus LOC could be defined as [11] 'A line of code is any line of program text that is not a comment or blank line regardless of the number of statements or
fragments of statements on the line. This specifically includes all lines containing program header, declaration and executable and non-executable statements.

This simple method also has some disadvantage, as a line of assembler is not the same as a line of COBOL. Measuring the software size in terms of LOC is like measuring a car stereo by the number of it's components rather than by the functions available.

4.2.2 FUNCTION POINTS

Any system can be seen from two viewpoints, the user's view and the developer's view. The size of the system can also be seen from the same two viewpoints, the user's functional perspective and developer's technical perspective. At the initial stages of the system, the functional view of the user is available. The perspective of the user relates to what the system can do for the user and the functional size supports the user's perspective. The developer's perspective deals with what needs to be built and this is the technical size measure. In the initial stages of the system, the functional view of the user is only available. Therefore, it is natural that software size estimation in the early life cycle typically uses functional size measure.
The function point measures functionality from the users point of view i.e., on the basis of what the user requests and receives in return from the system. Albrecht’s Function Point Analysis (FPA) decomposes into functional units viz., inputs, outputs, enquiries, internal logical files and external interface files. This approach is independent of the language, tools or methodologies used for implementation. Therefore software size estimation done early in the life cycle uses functional size measures.[71] International Functional point Analysis (FPA)[1,4] and Charles Symons Mark II Function Points [215] are two of the most widely used methods for obtaining functions points, a functional size measure.[183,193]

4.3 UNCERTAINTY IN ESTIMATION

One can perform size and hence cost estimation at any point in the software life cycle. It depends on the nature and characteristics of reliable information we have about the final product. There is a great deal of uncertainty about the actual specification. As we specify the system more fully and accurately, the uncertainty is reduced and more accurate estimates can be made. Traditionally used models need very accurate inputs and result in very precise values. They lead to a number of problems such as over commitment which has not been over come by these models. The reason for these problems lies in the lack of precision and vagueness present at various stages. In general it is considered that project managers can fairly readily specify independent variables in software metrics models using linguistic labels such as large level of project complexity and bigger
size of the software in the early stages of estimation. The uncertainty that prevails in the inputs to the models is captured by fuzzification and the imprecision of complexity is also managed by fuzzy logic. Instead of using crisp weights to define simple, average and complex measurement parameters, the proposed fuzzy model exploits overlapping amongst them. Such types of Fuzzy models provide considerable benefits in terms of reducing commitment, making full use of imprecise knowledge. Such type of model is proposed in this chapter.

4.4 FUNCTION POINT ANALYSIS

Function point analysis or FPA is a popular method for estimating size of software based on the functionality of the system. Albrecht [20, 22] proposed this model. Here the focus is to reflect the concerns of the user. Function points (FP) provide a level of abstraction between operational capability and programming language constructs. FP is an excellent measure of the functionality of the product [122,132,151,212]. According to IFPUG [207], the objective of function point analysis is to:

- Measure functionality of a software system as seen from the user’s perspective.
- Measure the size of software systems independent of technology used for implementation.
- Create a measurement methodology that is simple enough to minimize the overhead of measurement process.
• Create a consistent measure among various projects and organizations.

The schematic representation of the FPA method is given in Figure 4.1.

Fig 4.1: Schematic Representation of the FPA Method
Functionality factors in each of the five categories are counted and given weight by an expert. The product of the count and weights are computed and summed to produce an unadjusted function point count.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Function Type</th>
<th>Simple</th>
<th>Average</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Internal Logical files (ILJ)</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>External Interface files (EIF)</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>External inputs (EI)</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>External Outputs (EO)</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>External Queries (EQ)</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4.1: Complexity values of Function Types

The unadjusted function point (UFP) can be calculated as shown in Table 4.2 [8]
**WEIGHTING FACTOR**

<table>
<thead>
<tr>
<th>Measurement Parameter</th>
<th>count</th>
<th>simple</th>
<th>average</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of user Inputs</td>
<td>x</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Number of user Outputs</td>
<td>x</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Number of user Inquiries</td>
<td>x</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Number of files</td>
<td>x</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Number of external Interfaces</td>
<td>x</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

UFP

Table 4.2: Method to calculate UFP

Now, the total function point count is given as

\[ FPC = UFP \times VAF \]  \hspace{1cm} (4.1)

VAF, the value adjustment factor is given as

\[ VAF = (TDI \times 0.01) + 0.65 \]  \hspace{1cm} (4.2)

Where TDI is called the total degree of influence.

\[ FPC = UFP \times (TDI \times 0.01) + 0.65 \]  \hspace{1cm} (4.3)
The total degree of Influence (TDI) depends on 14 complexity adjustment factors[13] as given below:

1. Does the system require reliable backup and recovery?

2. Are data communications required?

3. Are there distributed processing functions required?

4. Is performance critical?

5. Will the system run in an existing, heavily utilized operational environment?

6. Does the system require on-line data entry?

7. Does the online data entry require the input transactions to be built over multiple screen or operations?

8. Are the master files updated on line?

9. Are the inputs, outputs, files or inquiries complex?

10. Is the internal processing complex?

11. Is the code designed to be reusable?

12. Are conversion and installation included in design?

13. Is the system designed for multiple installations and in different organizations?

14. Is the application designed to facilitate change and ease of use by the user?
Each of the factors is rated from 0-5 as below:

<table>
<thead>
<tr>
<th>No influence</th>
<th>Incidental</th>
<th>Moderate</th>
<th>Average</th>
<th>Significant</th>
<th>Essential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

An Expert rates these factors based on his intuition and experience.

Thus the function point calculation can be shown as [199]

$$\sum_{i=1}^{5} C_i \times W_i$$

**Figure 4.2: FPA Method**

EIF = External Interfaces Files.

ILF = Internal Logical files

EI = External Inputs

EO = External outputs
EQ = External Queries

TDI = Total Degree of Influence (It is the summation of the values of the 14 factors)

UFP = Unadjusted Function Points

This function point count is representative of the size of the software. Conversion from function point counts to actual line of code found in [207]

As example it approximately takes 125 lines of C code and 25 lines of fourth generation language to implement one function point.

4.5 FUZZY MODELING APPROACH

As can be seen each function type is classified as simple, average and complex as per table. It is measured using a rating scale of six linguistic values like simple, average, complex, etc. As can be seen if count of internal logic files is seven, it is simple, if it is 10, it is average. Then what if it is 9? This problem is caused by use of conventional quantization where the values are intervals. So the transition from one interval to another interval is abrupt rather than gradual.

Fuzzy sets are one of the methods which can be used here to mimic the human thought which can be applied here.[7,18,24,231,232] The advantage of quantization is[232]

1. Fuzzy logic mimics the way in which the humans interpret linguistic values.
2. The transition from one linguistic value to contiguous value is gradual rather than abrupt.

Fuzzy logic could also be applied to determine the effective rating and the fourteen complexity adjustments. Thus Fuzzy model for the functional point analysis method exploits the inherent fuzziness that is present in all model measurement parameters. (User inputs, user outputs, number of user inquiries, number of files, number of external interfaces and complexity factors). [90, 92]

The fuzzy model is shown as below in Figure 4.3.

![Diagram](image)

**Figure 4.3: The Fuzzy Model for FPA**

**Variables and their states for the model:**

**A. Count inputs:**

Our model has five count inputs namely:
1. Number of user inputs
2. Number of user outputs
3. Number of user inquiries
4. Number of files
5. Number of external interfaces

Each of the Inputs has three states. i.e. Simple, Average and Complex. The input states which are fuzzified are as shown below.

![Figure 4.4: Fuzzification of Internal Logical Files](image1)

![Figure 4.5: Fuzzification of External Interface Files](image2)
Figure 4.6: Fuzzification of External Inputs and External Outputs

Where,

$S =$ Simple

$A =$ Average

$C =$ Complex

B. Complexity Adjustment values.

The fourteen complexity adjustment values have been fuzzified as follows:

Figure 4.7: Fuzzification of Complexity Adjustment values
Fuzzy Membership function for Complexity Adjustment

NI = No Influence
I = Incidental
M = Moderate
A = Average
S = Significant
E = Essential

4.6 METHODOLOGY

The fuzzy model as shown in Fig 4.3 was used. All inputs namely number of user inputs, number of user outputs, number of user inquiries, number of files and number of external interfaces were fuzzified and given to this model. Defuzzification yields crisp values of Function Points.

4.7 CONCLUSION

In this chapter we have proposed a model of the Function Point Analysis method for size estimation in order to exploit the inherent fuzziness that is present in all the model measurements parameters. Instead of using crisp weights to define
simple, average and complex measurement parameters, the proposed fuzzy model
exploits overlapping amongst these. The proposed model has further fuzzified
complexity adjustment values. This leads to better size and hence cost estimates.