DEVELOPMENT

OF

COST ESTIMATION TOOL

MOHAMMAD ASIF

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India

DEVELOPMENT OF COST ESTIMATION TOOL

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Computer Science and Engineering

By

MOHAMMAD ASIF

Roll No. 109CS0630

Under supervision of

Prof. S. K. RATH

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, India.

May 09, 2013

Certificate

This is to certify that the thesis entitled ***Development of Cost Estimation Tool*** submitted by ***Mohammad Asif*** in the partial fulfillment of the requirements for the award of ***Bachelor of Technology Degree*** in ***Computer Science and Engineering*** at ***National Institute of Technology, Rourkela*** is an authentic work carried out by him under my supervision and guidance. To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university / institution for the award of any Degree or Degree.

Prof. S. K. Rath
Professor
CSE Department of NIT Rourkela

Acknowledgment

I am grateful to numerous local and global peers who have contributed towards shaping my project. At the outset, I would like to express my sincere thanks to Prof. S. K. Rath, for his advice during my project work. As my supervisor, he has constantly encouraged me to remain focused on achieving the goal. His inspiring guidance, valuable suggestion and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. He has helped me greatly and has been a source of knowledge.

I am thankful to all my friends who have patiently extended all sorts of help for accomplishing this undertaking. I sincerely thank everyone, who has provided us with inspirational words, a welcome ear, new ideas, constructive criticism, and their invaluable time.

I would like to thank the administrative and technical staff members of the department who have been kind enough to advise and help in their respective roles.

Last but not the least, I would like to dedicate this project to my family, for their love, patience and understanding.

Mohammad Asif

109CS0630

Abbreviations

- UML - Unified Modelling Language
- XML - Extensible Markup Language
- NEM - Number of External Methods
- NOA - Number of Attributes
- NSR - No of Service Requested
- NSR - No of Service Requested
- GUI - Graphical User Interface
- COCOMO - Constructive Cost Estimation Model
- SLOC - Source Lines of Code
- COCOMO - Constructive Cost Model
- FPA - Function Point Analysis
- DOI - Degree of Influence
- ILF - Internal Logical File
- EIF - External Interface File
- EI - External Input
- EO - External Output
- EQ - External Inquiry
- TUCP - Total Unadjusted Class Points
- ACP - Adjusted Class Points
- TDI - Total Degree of Influence
- TCF - Total Complexity Factor

Abstract

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. In software engineering, a class diagram in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. The aim of my project is to develop a tool to estimate the cost of a software using UML class diagram. This is achieved by converting UML class diagram to XML(Extensible Markup Language) representation. XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. By using the concept of class point approach, it calculates the total number of adjusted class point by parsing the XML file. First step for development of cost estimation tool requires understanding the concept of UML and XMI (XML Metadata Interchange). XMI is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML). The most common use of XMI is as an interchange format for UML models, although it can also be used for serialization of models of other languages. Conversion of UML class Diagram to XML representation using Magic Draw for parsing. Creating a XMI parser to find the NEM (Number of External Methods), NSR (Number of Service Requested) and NOA (Number of Attributes) and the type of classes. Using class point object oriented approach, calculate the effort required to develop a software system by NEM, NSR and NOA. Information procession size estimation includes identification and classification of classes, evaluation of complexity level of each class using 24 different type of drivers, estimation of the Total Unadjusted Class Point and estimation of technical complexity factor estimation. After all these calculation we can calculate Final class point evaluation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A survey says, almost one-third projects exceed their budget and is delivered late and two-thirds of all projects overrun their original estimates. It is impossible for a manager or system analyst to accurately predict the cost and effort required to develop a software. Without accurate cost estimation capability, project managers can't determine how much time and manpower the project should take and that means the software portion of the project is out of control from its beginning.
It is difficult to understand and estimate a software product that cant be seen and touched. Software grow and change when it is written. In project management, the most challenging task is cost estimation. It is necessary to correctly estimate required resources and schedules for software development projects. Software cost estimation process includes the following:

- Estimation of the size of the software product to be produced
- Estimation of the effort required
- Development of preliminary project schedules
- Estimation of overall cost of the project

First step in any estimate is to understand and define the system to be estimated. A software is invisible, intractable, and intangible. It is inherently more difficult to understand and estimate a product or process that cannot be seen and touched. When hardware design is inadequate, or when it fails to perform, the solution is often attempted through changes to the software. Changes may occur late during the development process, which results in unanticipated software growth. After twenty years research, many software cost estimation methods are available like, estimation by analogy, expert judgment method, algorithmic methods, bottom-up method, and top-down method. No method is necessarily better or worse than the other, in fact, their weaknesses and strengths are often complimentary. It is very important to understand the strengths and weaknesses of every method when you want to estimate your projects.

1.1 Top Down Approach

In top-down approach, breaking down the system to gain insight into its compositional sub-systems. An overview of the system is formulated in this approach, specifying but not detailing any first-level subsystems. Each subsystem is again refined in greater detail. Top-down estimating method is also named as Macro Model. Using this method, an overall cost estimation for a project is derived using the global properties of the software project, and then the project is divided into various low-level components. Putnam model is developed using this approach. When global properties are known, this method is most applicable for early cost estimation. In early phase of software development, it is very useful because there are no detailed information available.

1.1.1 Putnam Model

The Putnam model is an empirical software effort estimation model. It describes the time and effort required to finish a software project of specified size. It is one of the earliest model developed, and is among the most widely used. It is very sensitive to the development time: decreasing the development time can greatly increase the person-months needed for development. Using Putnam model, SLIM [1] tool is developed.Man month required for development is given by the formula [2]:
Technical constant C= $size * B^{1/3} * T^{4/3}$
Total Person Months B=$(1/T)^4 * (size/C)^3$
T= Required Development Time (in years)

Size is estimated in LOC

Where,

C: A parameter determined on the basis of historical data of the past projects and dependent on the development environment.

Rating: C=2,000 (poor), C=8000 (good) C=12,000 (excellent).

1.2 Bottom Up Approach

A bottom-up approach is the piecing together of systems to give rise to grander systems. It emphasize mainly on coding and early testing, which begins as soon as the first module has been specified. This approach, runs the risk that modules may be coded without having a clear idea of how they link to other parts of the system, and such linking may not be as easy as first thought. Main benefit of the bottom-up approach is re-usability of code. Using bottom-up estimating method, cost of each software components is estimated and then combine all the results to arrive at an estimated cost of overall project. It aims at constructing the estimate of a system from the knowledge accumulated from the small software components and their interactions. COCOMO model is developed using this approach.

1.2.1 COCOMO Model

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model. It is a regression model which uses basis regression formula with parameters that are derived from historical project data and current as well as future project characteristics. Software development effort is calculated in terms of program size by COCOMO. Program size is estimated in thousands of source lines of code (SLOC). COCOMO assumes that the system and software requirements have been defined already, and that these requirements are stable.

The COCOMO model has a very simple form [3]:

MAN-MONTHS = $K1 * (ThousandsofDeliveredSourceInstructions)^{K2}$

Here K1 and K2 are two parameters dependent on the application and development environment.

Chapter 2

LITERATURE REVIEW

In software project development, size evaluation is one of the main tasks with reliable cost and effort estimations. To estimate the size of a software system several measures have been defined so far. Some are as follows:

- Function point Approach
- Class Point Approach

2.1 FUNCTION POINT APPROACH

Function Point Analysis (FPA) [4] was developed by IBM in response to a number of problems arising in measuring the size of system in terms of lines of codes. FPA measures size of an application system in two areas: the specific user functionality and the system characteristics. The specific user functionality, is the measurement of functionality delivered by the application for user request. The five function types identified are: external output, external enquiries, external input, external interface files and internal logical files. For each function identified as one of the five function types given, it is further classified as low, average or high and a weight is given to each. The sum of weights tells about the size of information processing and is referred as Unadjusted Function Points.
Function Point = (User Functionality) * (System Characteristics)
The general functionality of systems will be affected by some system characteristics. To rate general functionalities of the system, fourteen general system characteristics are identified. Degree of Influence (DI) ranges from 0-5, from no influence to strong

influence, is determined for each of the general system characteristics. The sum of all these degree of influence will in turn determine a Value Adjustment Factor for the whole projects.
The product of the Value Adjustment Factor and Unadjusted Function Point gives the size of the application expressed in term of Adjusted Function Point [5].
Adjusted Function Point = $(UnadjustedFunctionPoint) * (ValueAdjustmentFactor)$
The important steps of Function Point Analysis are [6]:

- Determine the type of function point,
- Identify application boundary,
- Determine unadjusted function point,
- Determine value adjustment factor,
- Calculate final adjusted function point.

2.1.1 Determine the type of function point count

FPA technique applies different formula while measuring size of system for software development and maintenance. So, the type of function point count should be determined at the outset. Three types of function point counts [7]:

1. Enhancement project function point count
2. Development project function point count
3. Application function point count

2.1.2 Identify application boundry

An application boundary, which defines a system viewed by the users and determines any interaction with other systems, should be determined first so as to set up the scope for the related functions to be identified.

2.1.3 Determine the Unadjusted Function Point

The unadjusted function point reflect the functionality of logical system provided to the user. Five function types are used to determine unadjusted function point. Those function points are:

- Internal Logical File (ILF)
- External Interface File (EIF)
- External Input (EI)
- External Output (EO)
- External Inquiry (EQ)

Each function type is assessed for its complexity (low, average or high) as follows:

- Depending on the number of file type referenced (FTR) and data element type (DET), EI, EO and EQ are given complexity ratings; and
- Depending on number of record element types (RET) and data element types (DET), EIF and ILF are given complexity rating [8].

2.1.4 Determine the Adjustment Factor Value

There are in total fourteen general system characteristics which account for the overall influences that will affect the complexity and size of the system to be provided to the users. These include:

Table 2.1: Table for calculating complexity using Function Point

Function Type	Complexity		
	LOW	AVERAGE	HIGH
External Input(EI)	3	4	6
External Output (EO)	4	5	7
External Inquiry (EQ)	3	4	6
External Interface File(EIF)	5	7	10
Internal Logical File (ILF)	7	10	15

1. Data Communication
2. Distributed Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. On-line Data Entry
7. End-User Efficiency
8. On-line Update
9. Complex Processing
10. Reusable Code
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Ease of Change

Each general system characteristics vary from 0-5 to show its degree of influence. The values of DI represents:

1. 0 = No influence when present
2. 1 = Insignificant influence
3. 2 = Moderate influence
4. 3 = Average influence
5. 4 = Significant influence
6. 5 = Strong influence

The DI factor will range from 0.65 to 1.35.

2.1.5 Calculation of Final Adjusted Function Point

After determining the adjustment factor and unadjusted function points, the adjusted function points, can be obtained by multiplying the two figures.

2.2 CLASS POINT APPROACH

Class Point approach provides a system level estimation of the size of Object Oriented products. It has been derived by recasting the ideas underlying the function point analysis within the Object Oriented paradigm and by suitably combining well-known OO measures. The process of Class Point size estimation is composed of 3 main phases, corresponding to analogous phases in function point approach. The following steps shows how class point is calculated:

- Information procession size estimation
  1. Identification and classification of class
  2. Calculation of complexity level of each class
  3. Calculating Total Unadjusted Class Points
- Estimation of Technical Complexity Factor(TCF)
- Final evaluation of Class Point

2.2.1 Identification and Classification of User Classes

In order to classify a class, four types of system component is defined which are named as Problem Domain Type (PDT), Human Interaction Type (HIT), Data Management Type (DMT) and Task Management Type (TMT).

2.2.2 Evaluation of a Class Complexity Level

Complexity level of each class is calculated using two approaches named as CP1 and CP2. NEM, NSR and NOA are required to calculate CP1 and CP2 values. The number of external methods (NEM) measures size of the interface of a class and is calculated by the number of locally defined public methods. The number of service requested (NSR) provides measure of the interconnection of system components. The number of attributes is an additional parameter. In CP1 we consider NEM and NSR and in CP2 all the three are taken into account.

Table 2.2: Table Complexity level evaluation for CP1

	0-4 NEM	5-8 NEM	>=9 NEM
0-1 NSR	LOW	LOW	AVERAGE
2-3 NSR	LOW	AVERAGE	HIGH
>=4 NSR	AVERAGE	HIGH	HIGH

Table 2.3: Table Complexity level evaluation for CP2

0-2 NSR	0-5 NOA	6-9 NOA	>=10 NOA
0-4 NEM	LOW	LOW	AVERAGE
5-8 NEM	LOW	AVERAGE	HIGH
>=9 NEM	AVERAGE	HIGH	HIGH

3-4 NSR	0-4 NOA	5-8 NOA	>=9 NOA
0-3 NEM	LOW	LOW	AVERAGE
4-7 NEM	LOW	AVERAGE	HIGH
>=8 NEM	AVERAGE	HIGH	HIGH

>=5 NSR	0-3 NOA	4-7 NOA	>=8 NOA
0-2 NEM	LOW	LOW	AVERAGE
3-6 NEM	LOW	AVERAGE	HIGH
>=7 NEM	AVERAGE	HIGH	HIGH

2.2.3 Estimating the Total Unadjusted Class Point

Once the complexity of each class has been calculated, we can calculate Total Unadjusted Class Point (TUCP). To calculate TUCP one needs to fill the TUCP table. Each entry in the table expresses the weighted number of classes whose complexity level and typology are given by the corresponding row and column [9].

$$TUCP = \sum_{i=1}^{4} \sum_{j=1}^{3} W_{ij} A_{ij}$$

Table 2.4: Table for calculating TUCP using Class Point

System Complexity Type	Complexity		
	LOW	AVERAGE	HIGH
Problem Domain Type(PDT)	...*3	...*6	...*10
Human Interaction Type(HIT)	...*4	...*7	...*12
Data Management Type(DMT)	...*5	...*8	...*13
Task Management Type(TMT)	...*4	...*6	...*9
TUCP	Total Unadjusted Class Point		

2.2.4 Technical Complexity Factor Estimation

The technical complexity factor is calculated by assigning the degree of influence ranging from 0 to 5 that 18 general system characteristics have on application. The sum of influence degrees related to a general characteristics of a system form Total Degree of Influence (TDI). Now, TCF can be calculated using the formula:

$$\text{TCF} = 0.55 + 0.01 * \text{TDI}$$

2.2.5 Calculation of Adjusted Class Point

Adjusted class point can be calculated using the TUCP value and TCF value. The formulae to calculate adjusted class point is:

$$\text{ACP} = \text{TUCP} * \text{TCF}$$

Chapter 3

Proposed Work

3.1 Concepts Used

3.1.1 UML Diagrams

Unified Modeling Language (UML) is a standardized, general-purpose modeling language in the field of software engineering. UML can be described as a general purpose visual modeling language to construct, visualize, specify, and document a software system. It is a pictorial language used to make software blue prints. There are two main categories of diagrams and then they are again divided into sub-categories:

- Structural Diagram
- Behavioural Diagram

Structural Diagram

The structural diagrams represent the static aspect of system. The static aspect represent those parts of a diagram which forms the main structure and are therefore stable. Static parts are represents by objects, components, classes, interfaces, and nodes. The four structural diagrams are:

- Class Diagram
- Object Diagram
- Component Diagram

- Deployment Diagram

Behavioral Diagrams

Behavioral diagrams capture the dynamic aspect of system. Dynamic aspect can be further described as the changing parts of a system. UML diagrams has the following five types of behavioral diagrams:

- Use Case Diagram
- Collaboration Diagram
- Activity Diagram
- State Chart Diagram
- Sequence Diagram

Class Diagram

Class diagrams is the most common diagram used in UML. Class diagram consists of interfaces, classes, collaboration and associations. Class diagrams represent the object oriented view of system which is static in nature. Active class is used in class diagram to represent concurrency of the system. In class diagram every class member has a visibility. Visibility are of different types like, + Public, - Private, '#' Protected. Between two classes there exist a relationship. Aggregation, association and composition are types of relationship which exist between classes.

3.2 Proposed Work

Development of a cost estimation tool using extended class point approach. The steps followed for the development of tool are shown in figure 3.1 below

3.2.1 Draw UML class Diagram

UML class diagram shows different classes with their attributes, method and the relationship between them. Example of UML class diagram is shown in figure 3.2 below [10].

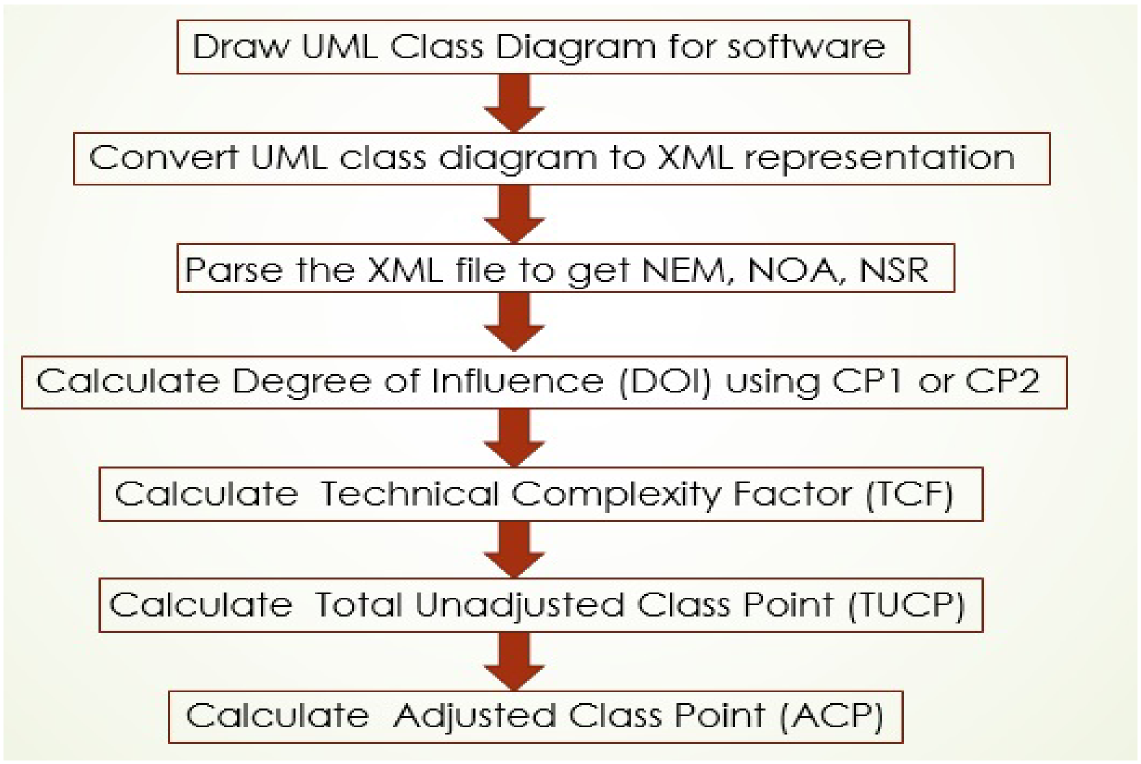

Draw UML Class Diagram for software
Convert UML class diagram to XML representation
Parse the XML file to get NEM, NOA, NSR
Calculate Degree of Influence (DOI) using CP1 or CP2
Calculate Technical Complexity Factor (TCF)
Calculate Total Unadjusted Class Point (TUCP)
Calculate Adjusted Class Point (ACP)

Figure 3.1: Steps followed for calculation of ACP

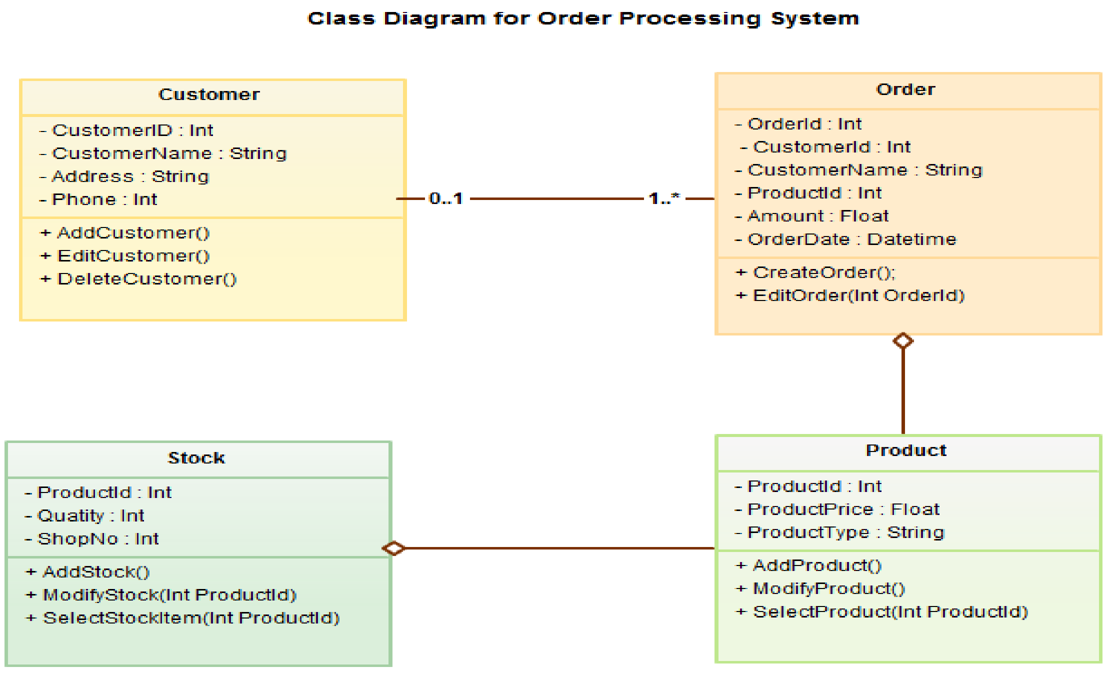


Figure 3.2: Class Diagram

3.2.2 Conversion of Class Diagram to XML representation

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding a documents in a format that is both machine-readable and human-readable. Several schema systems exist to aid the definition of XML based languages and many application programming interfaces (APIs) have been developed to aid software developers with processing XML data. Class diagram is converted to XML representation using software like MagicDraw. The output generated from MagicDraw shows all the class with its access modifiers, number of attributes, number of methods, and type of relationship which exist between every class. A sample output obtained from MagicDraw for a single class is represented like this as shown in figure 3.3 below.

```
{<packagedElement xmi:type='uml:Class' xmi:id=
    '_17_0_3_ef2031a_1363700997451_19172_3106' name='ATM'>
<generalization xmi:type='uml:Generalization' xmi:id=
    '_17_0_3_ef2031a_1363701013010_817561_3148' general=
    '_17_0_3_ef2031a_1363701003171_605450_3127'/>
<ownedAttribute xmi:type='uml:Property' xmi:id=
    '_17_0_3_ef2031a_1363701110626_186107_3152' name=
    'atmID' visibility='private'/>
<ownedAttribute xmi:type='uml:Property' xmi:id=
    '_17_0_3_ef2031a_1363701139314_328886_3154' name=
    'loc' visibility='private'/>
<ownedAttribute xmi:type='uml:Property' xmi:id=
    '_17_0_3_ef2031a_1363701144158_731856_3156' name=
    'state' visibility='private'/>
<ownedOperation xmi:type='uml:Operation' xmi:id=
    '_17_0_3_ef2031a_1363701172398_685293_3161' name=
    'shutdown' visibility='public'/>
<ownedOperation xmi:type='uml:Operation' xmi:id=
    '_17_0_3_ef2031a_1363701177937_804194_3164' name=
    'createsession' visibility='public'/>
<ownedOperation xmi:type='uml:Operation' xmi:id=
    '_17_0_3_ef2031a_1363701182570_354796_3167' name=
    'Login' visibility='public'/>
</packagedElement>
```

Generalization represent the type of relation between classes, ownedAttribute represent the attribute of the class and ownedOperation represent the methods of the class.

3.2.3 Parsing XML file

The XML file obtained from the class diagram is then parsed to get the relevant information required for cost estimation using class point approach. By parsing the XML file we get the type of class, relationship between the class, NEM, NOA and NSR. The information obtained from parsing is used to calculate the adjusted class point value using CP1 or CP2 approach.

3.2.4 Calculation of Degree of Influence (TDI)

The Adjustment Factor (AF) is calculated for the system using approach CP1 or CP2. In CP1 we consider 22 general system characteristics and in CP2, twenty four general characteristics are considered. All general system characteristics are assigned values ranging from 0 to 5 depending on how they influence the system. It is decided according to designers point of view. The DI values are:

- 0 - No influence or not present
- 1 - Insignificant influence
- 2 - Moderate influence
- 3 - Average influence
- 4 - Significant influence
- 5 - Strong influence

After assigning degree of influence values to all system characteristics the AF is calculated. TDI is the sum of all degree of influence values of system characteristics.

3.2.5 Calculation of Adjustment Factor

The AF value is dependent of the value of TDI and is calculated using the formula:

$$AF = 0.55 + 0.01 * TDI$$

3.2.6 Evaluation of Class Complexity Level

In extended class point approach complexity level of each class is calculated using same two approaches named as CP1 and CP2 as it was in class point approach. The only difference is that here we have divided the complexity level into four i.e. Low, Average, High and Very. All other process is same as in class point approach. The table for complexity level evaluation is given below [11]

Table 3.1: Table Complexity level evaluation for CP1

	0-4 NEM	5-8 NEM	9-12 NEM	>=13 NEM
0-1 NSR	LOW	LOW	AVERAGE	HIGH
2-3 NSR	LOW	AVERAGE	HIGH	HIGH
4-5 NSR	AVERAGE	HIGH	HIGH	VERY HIGH
>=6 NSR	HIGH	HIGH	VERY HIGH	VERY HIGH

Table 3.2: Table Complexity level evaluation for CP2

0-2 NSR	0-5 NOA	6-9 NOA	10-14 NOA	>=15 NOA
0-4 NEM	LOW	LOW	AVERAGE	HIGH
5-8 NEM	LOW	AVERAGE	HIGH	HIGH
9-12 NEM	AVERAGE	HIGH	HIGH	VERY HIGH
>=13 NEM	HIGH	HIGH	VERY HIGH	VERY HIGH

3-4 NSR	0-4 NOA	5-8 NOA	9-13 NOA	>=14 NOA
0-3 NEM	LOW	LOW	AVERAGE	HIGH
4-7 NEM	LOW	AVERAGE	HIGH	HIGH
8-11 NEM	AVERAGE	HIGH	HIGH	VERY HIGH
>=12 NEM	HIGH	HIGH	VERY HIGH	VERY HIGH

>=5 NSR	0-3 NOA	4-7 NOA	8-12 NOA	>=13 NOA
0-2 NEM	LOW	LOW	AVERAGE	HIGH
3-6 NEM	LOW	AVERAGE	HIGH	HIGH
7-10 NEM	AVERAGE	HIGH	HIGH	VERY HIGH
>=11 NEM	HIGH	HIGH	VERY HIGH	VERY HIGH

3.2.7 Calculation of Total Unadjusted Class Point

Once the complexity of each class has been calculated, we can calculate Total Unadjusted Class Point (TUCP). To calculate TUCP one needs to fill the TUCP table. Each entry in the table expresses the weighted number of classes whose complexity level and typology are given by the corresponding row and column. Table 3.3 [12] shows how the TUCP is calculated. TUCP is calculated using the formula:

$$TUCP = \sum_{i=1}^{4} \sum_{j=1}^{4} W_{ij} A_{ij}$$

Table 3.3: Table for calculating TUCP using Extended Class Point

System Complexity Type	Complexity			
	LOW	AVERAGE	HIGH	VERY HIGH
Problem Domain Type(PDT)	...*3	...*6	...*10	...*15
Human Interaction Type(HIT)	...*4	...*7	...*12	...*19
Data Management Type(DMT)	...*5	...*8	...*13	...*20
Task Management Type(TMT)	...*4	...*6	...*9	...*13
TUCP	Total Unadjusted Class Point			

3.2.8 Calculation of Adjusted Class Point

Finally, after calculating TUCP value and AF value, we can calculate the value of ACP. The formula for calculating ACP is:

ACP = TUCP * AF

After the ACP is calculated we can easily calculate the effort required to develop the system/software. The methods through which we can calculate the effort from ACP value are:

- Multivariate Adaptive Regression Splines (MRS)
- K Nearest Neighbor Regression (KNN)
- Multi-Layer Perceptron (MLP)
- Projection Pursuit Regression (PPR)

Through regression testing, effort required for development of software is calculated [13].

Chapter 4

GRAPHICAL USER INTERFACE

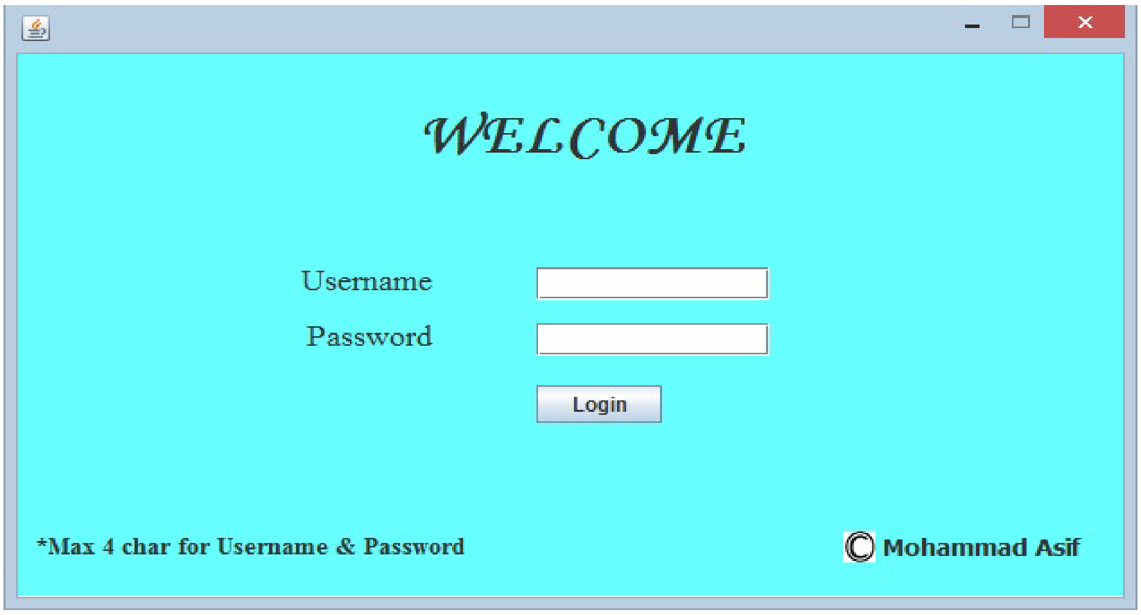


Figure 4.1: Login Form

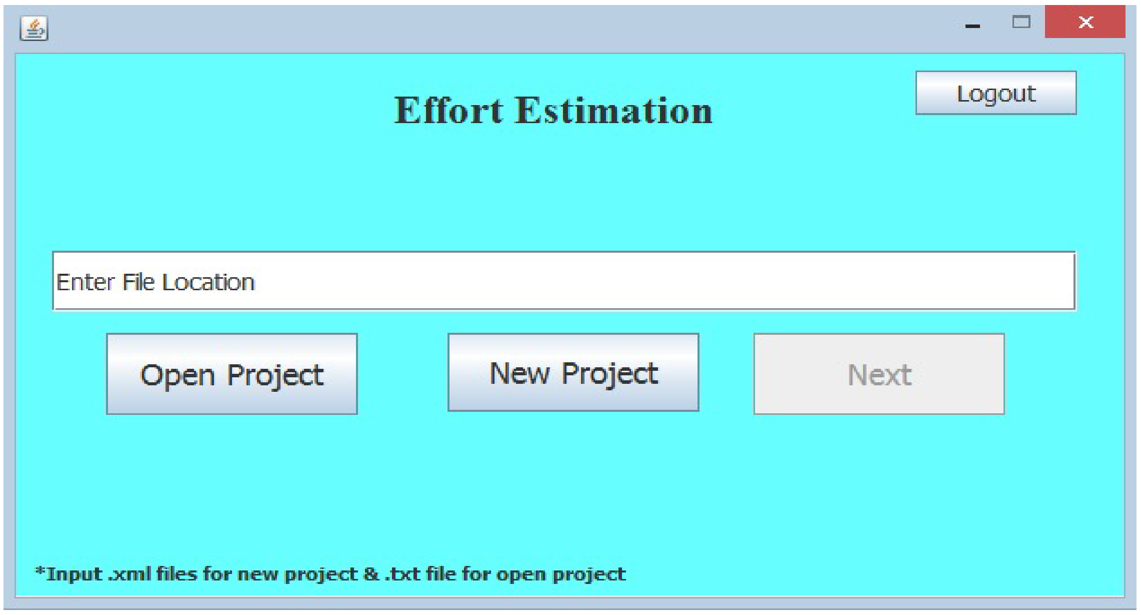
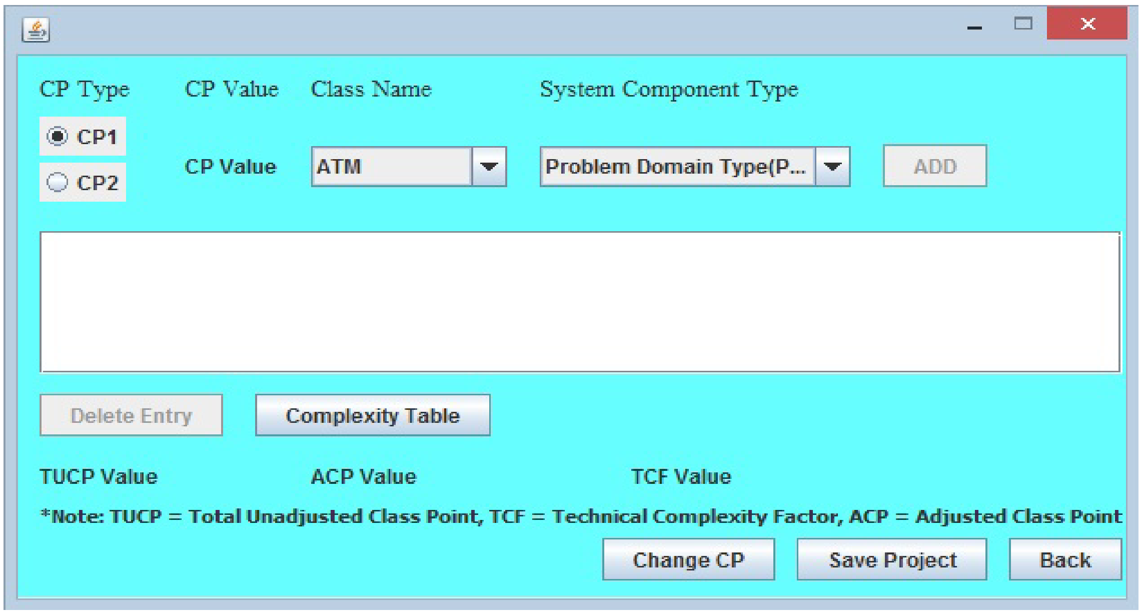


Figure 4.2: Path of XML file



CP Type CP Value Class Name System Component Type

[✓ CP1] CP2

CP Value | ATM | Problem Domain Type(P... | ADD

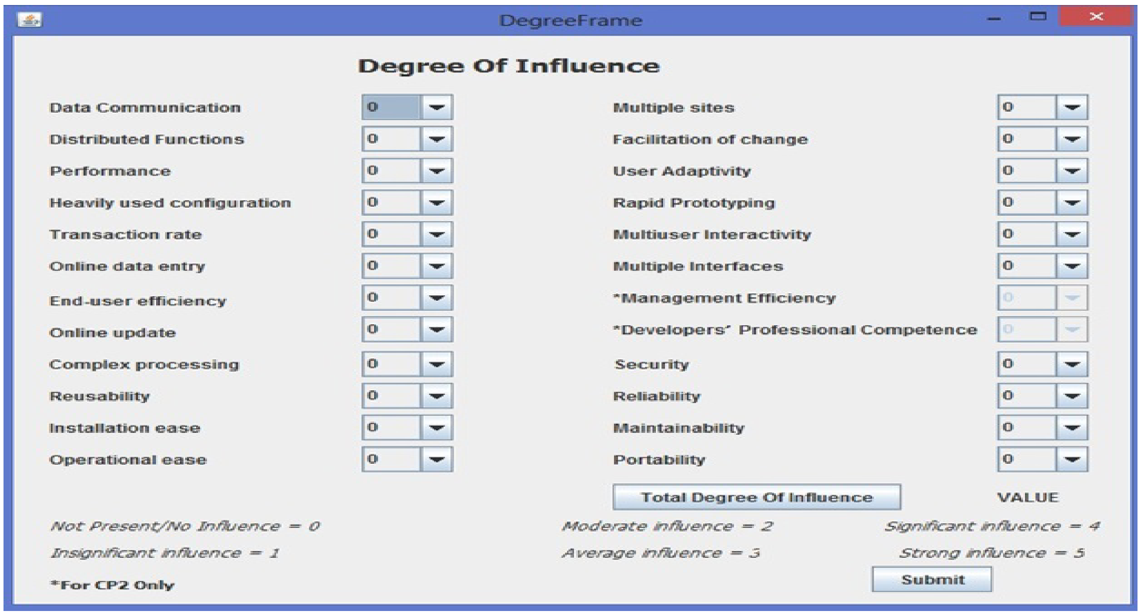
Delete Entry | Complexity Table

TUCP Value ACP Value TCF Value

*Note: TUCP = Total Unadjusted Class Point, TCF = Technical Complexity Factor, ACP = Adjusted Class Point

Change CP | Save Project | Back

Figure 4.3: Main Form



DegreeFrame

Degree Of Influence

Data Communication	0	Multiple sites	0
Distributed Functions	0	Facilitation of change	0
Performance	0	User Adaptivity	0
Heavily used configuration	0	Rapid Prototyping	0
Transaction rate	0	Multiuser Interactivity	0
Online data entry	0	Multiple Interfaces	0
End-user efficiency	0	*Management Efficiency	0
Online update	0	*Developers' Professional Competence	0
Complex processing	0	Security	0
Reusability	0	Reliability	0
Installation ease	0	Maintainability	0
Operational ease	0	Portability	0

Total Degree Of Influence | VALUE

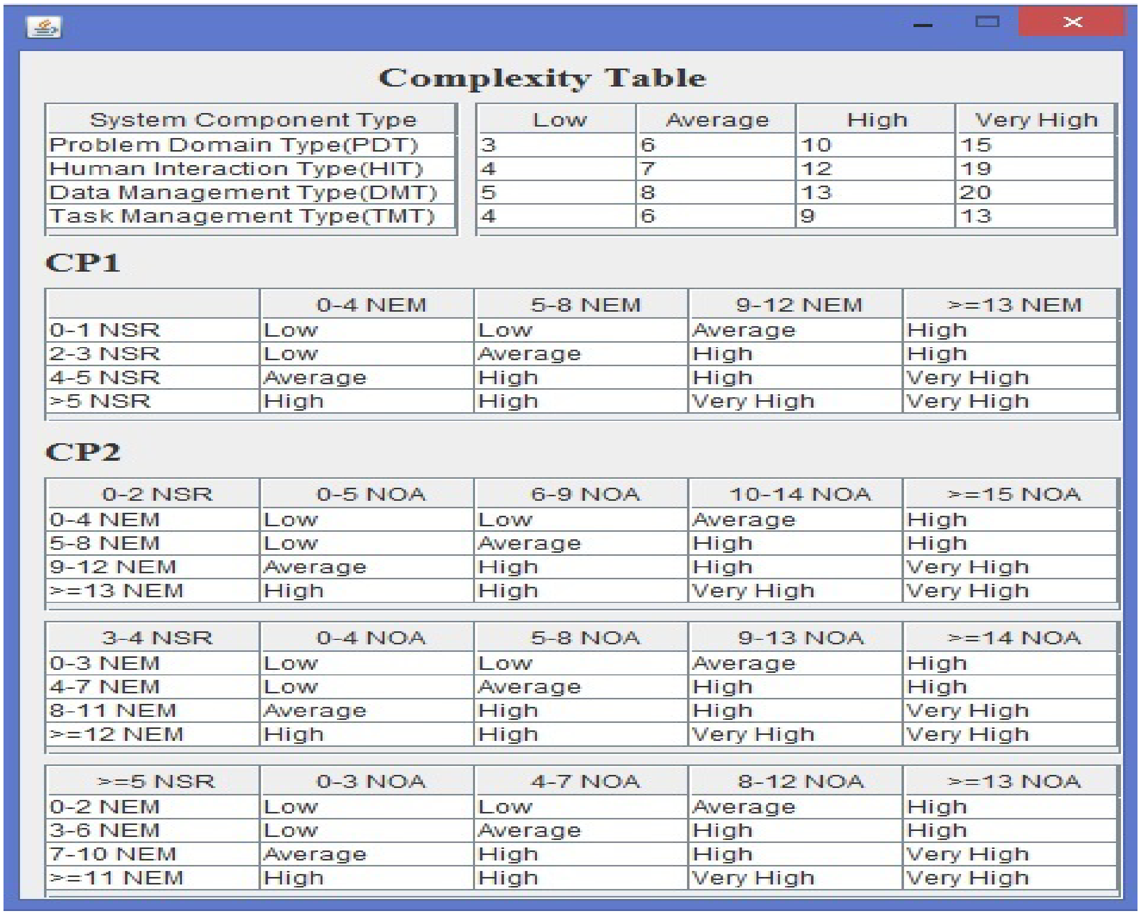
Not Present/No Influence = 0 *Moderate influence = 2* *Significant influence = 4*

Insignificant influence = 1 *Average influence = 3* *Strong influence = 5*

*For CP2 Only

Submit

Figure 4.4: TDI Form



Complexity Table

System Component Type	Low	Average	High	Very High
Problem Domain Type(PDT)	3	6	10	15
Human Interaction Type(HIT)	4	7	12	19
Data Management Type(DMT)	5	8	13	20
Task Management Type(TMT)	4	6	9	13

CP1

	0-4 NEM	5-8 NEM	9-12 NEM	>=13 NEM
0-1 NSR	Low	Low	Average	High
2-3 NSR	Low	Average	High	High
4-5 NSR	Average	High	High	Very High
>5 NSR	High	High	Very High	Very High

CP2

0-2 NSR	0-5 NOA	6-9 NOA	10-14 NOA	>=15 NOA
0-4 NEM	Low	Low	Average	High
5-8 NEM	Low	Average	High	High
9-12 NEM	Average	High	High	Very High
>=13 NEM	High	High	Very High	Very High

3-4 NSR	0-4 NOA	5-8 NOA	9-13 NOA	>=14 NOA
0-3 NEM	Low	Low	Average	High
4-7 NEM	Low	Average	High	High
8-11 NEM	Average	High	High	Very High
>=12 NEM	High	High	Very High	Very High

>=5 NSR	0-3 NOA	4-7 NOA	8-12 NOA	>=13 NOA
0-2 NEM	Low	Low	Average	High
3-6 NEM	Low	Average	High	High
7-10 NEM	Average	High	High	Very High
>=11 NEM	High	High	Very High	Very High

Figure 4.5: Complexity Form

Chapter 5

Simulation and Results

Applying regression technique over forty data sets [9] of CP1 and CP2. The effort is calculated as shown in the below diagrams.
Figure 5.1 shows the relation between input data and effort calculated using CP1 Approach.
Figure 5.2 shows the relation between input data and effort calculated using CP2 Approach.

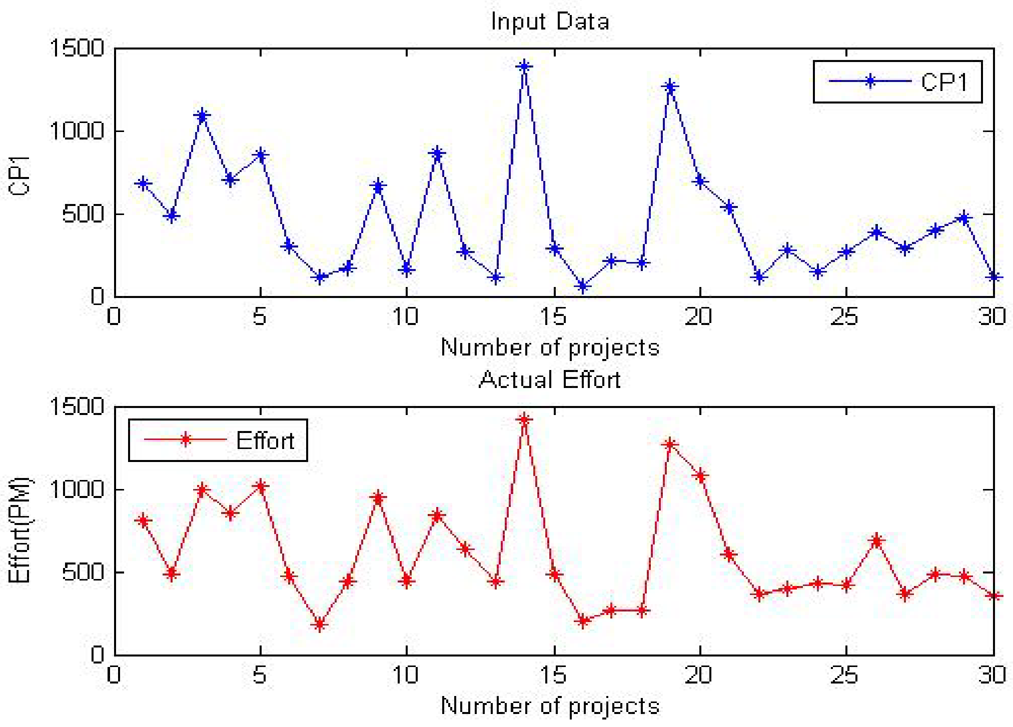


Figure 5.1: Fig. Effort Calculation Using CP1

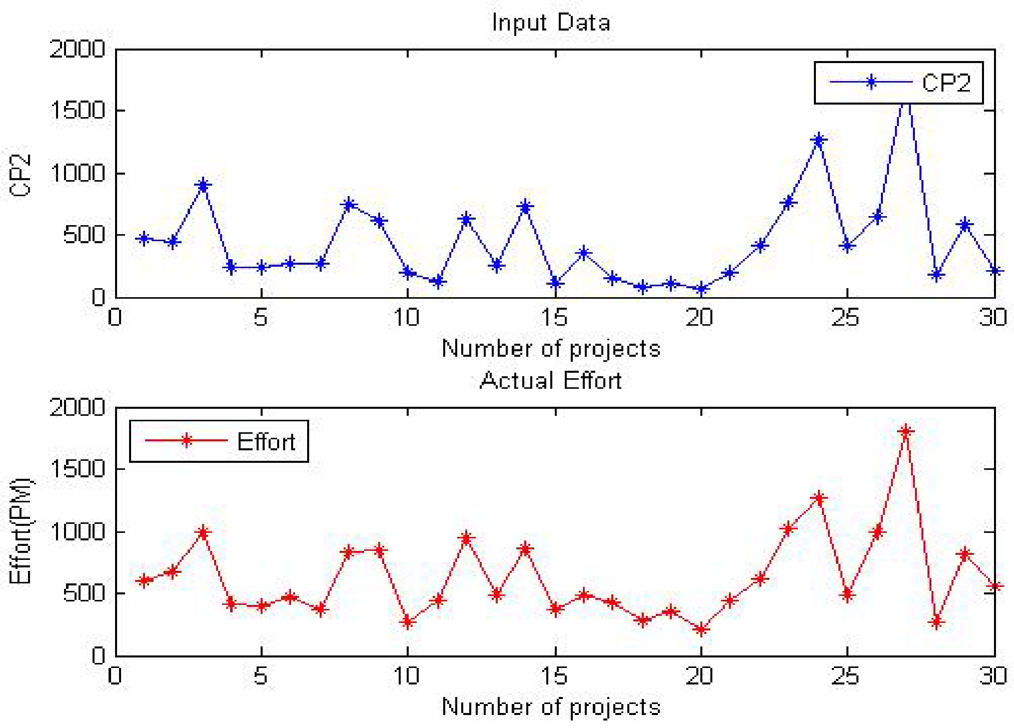


Figure 5.2: Fig. Effort Calculation Using CP2

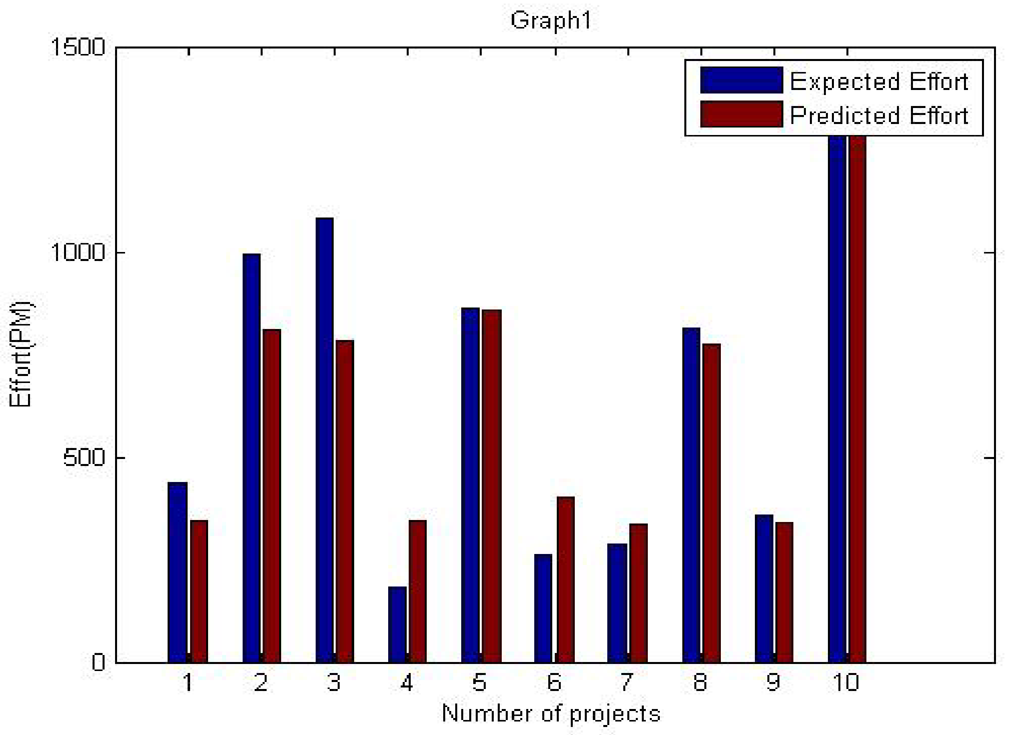


Figure 5.3: Fig. Result obtained after regression

Chapter 6

Conclusions

The Extended Class Point Approach provides system-level size estimation of Object Oriented product and from empirical validation, it exhibits better performance than the Class Point Approach. Software developed for class point calculation is simple to use. Calculating Adjusting Class Point value for large number of softwares using this tool, compared the actual effort and the estimated effort using regression analysis. Through the effort estimation, we can conclude that, how much effort the project has used and then we can have a deeper knowledge of developer teams professional skill level. Leaders of company need this kind of data to manage the company and arrange tasks according to the developer's professional skill.

Bibliography

[1] Nikki Panlilio-Yap. Software estimation using the slim tool. In *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research - Volume 1*, CASCON '92, pages 439–475. IBM Press, 1992.

[2] K. Pillai and V.S. Sukumaran Nair. A model for software development effort and cost estimation. *Software Engineering, IEEE Transactions on*, 23(8):485–497, 1997.

[3] Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, and Bert Steece. *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[4] T. Uemura, S. Kusumoto, and K. Inoue. Function point measurement tool for uml design specification. In *Software Metrics Symposium, 1999. Proceedings. Sixth International*, pages 62–69, 1999.

[5] CPM IFPUG. Counting practices manual, release 4.1. 1. *IFPUG–International Function Point Users Group*, 2000.

[6] T. Fetcke, A. Abran, and Tho-Hau Nguyen. Mapping the oo-jacobson approach into function point analysis. In *Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23. Proceedings*, pages 192–202, 1997.

[7] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1):33–53, 2007.

[8] J.E. Matson, B.E. Barrett, and J.M. Mellichamp. Software development cost estimation using function points. *Software Engineering, IEEE Transactions on*, 20(4):275–287, 1994.

[9] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello. Class point: an approach for the size estimation of object-oriented systems. *Software Engineering, IEEE Transactions on*, 31(1):52–74, 2005.

[10] SangEun Kim, William Lively, and Dick Simmons. An effort estimation by uml points in early stage of software development. In *Software Engineering Research and Practice'06*, pages 415–421, 2006.

[11] Wei Zhou and Qiang Liu. Extended class point approach of size estimation for oo product. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 4, pages V4–117–V4–122, 2010.

[12] S. Kanmani, J. Kathiravan, S. Senthil Kumar, and M. Shanmugam. Neural network based effort estimation using class points for oo systems. In *Proceedings of the International Conference on Computing: Theory and Applications*, ICCTA '07, pages 261–266, Washington, DC, USA, 2007. IEEE Computer Society.

[13] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *Software Engineering, IEEE Transactions on*, 23(11):736–743, 1997.