ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Result Paper on a Tool to Evaluate the Performance of UCP

Dr. Bikrampal Kaur[1], Ms. Dipeeka Rani Bathla[2]
[1]Professor, Department of CSE
Chandigarh Engineering College, Mohali, Punjab, India
[2]Research Scholar, Department of CSE
Chandigarh Engineering College, Mohali, Punjab, India
mca.bikram@gmail.com

ABSTRACTS

Estimates of cost and schedule in software projects are based on a prediction of the size of the future system. Unfortunately, the estimation of cost and schedule rarely comes within budget and on-time. Traditional cost models may take software size as an input parameter, and then apply a set of adjustment factors or 'cost drivers' to compute an estimate of total effort. In Object oriented software production, use case describes functional requirements. The use case model is used to predict the size of the future software system at an early development stage. This dissertation work describes the various estimation methods combined into a single tool. Tools consists estimation models of main input parameter size. These are Source Lines Of Code (SLOC) and Function Point (FP). This Size Input is incorporated into cost estimation models like COCOMO I and COCOMO II from where we estimate other factors like Effort, duration, Cost etc. But the main aim of this tool is to provide better estimates of size for this time trend languages which are based on Object Oriented Paradigm.

Keywords: Use Case Point (UCP), FP, COCOMO, TCF, EF, UUCW, UUCP, UML, UAW, LOC.

INTRODUCTION

Today, software are based on object oriented paradigm and OOP languages. Developers use Unified Modeling Language (UML) notations and diagrams for estimation of each aspect of software development. So the main factor is to use such a technique that supports the OOPs for estimation purposes. Main approach of this thesis is to implement Use Case Point estimation (UCP) Technique to overcome the drawbacks of FP which was considered as a procedural oriented (POP). The steps in the use case point's method as used by Karner : First, categorize the actors in the use case model as simple, average or complex and calculate the total **unadjusted actor weight (UAW)** by counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the points. Next, categorize the use cases as simple, average or complex, depending on the number of transactions, including the transactions in alternative flows. Then the **unadjusted use case weights (UUCW)** are calculated by counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The **UAW** is added to the **UUCW** to get the **unadjusted use case points (UUCP)**. Next, the use case points are adjusted based on the values assigned to a number of technical factors and environmental factors. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the overall project.

COCOMO I and COCOMO II

Non-Proprietary Models: Models that are fully documented in public domains are called Non-Proprietary Models. We have sufficient Information about the models that comes under this category.

Examples of Non-Proprietary are: COCOMOI, COCOMOII.

COCOMOI (Constructive Cost Model): Original COCOMO stands for Constructive cost model. The word "constructive" implies the openness of the model because while using COCOMO model for estimation we can easily show the reasons of estimated results. The model was first introduced by Dr. Barry Boehm in 1981, Developed at TRW, a US defense contractor. The Constructive Cost Model (COCOMO) is an algorithmic Software

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

cost model that uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics. A survey on 63 projects ranging from 2,000 to 10,000 lines of code, and programming languages ranging from assembly to PL/I were conducted and on the basis of their mutual outputs, the model was framed and the name given to that model was COCOMO I. COCOMO estimates rely on waterfall model. COCOMO I is a simple on-line available cost estimation model which is used for the estimation of efforts or person-months required to develop software. Other factors like duration, productivity, cost estimations are also can be derived through COCOMO I.

COCOMO I depend on two main equations :

Development Effort: MM= a*KDSI^b

Based on MM – Man-Months/Person-months/Staff months is one month of effort by one person.

Duration: TDEV=2.5*MM^c

The coefficients a, b and c are constants and their values depend on the mode of the development.

Modes of development are :

- **Organic Mode**: Typical 2-50 KLOC sized projects come in this mode category. Usually a small team of experienced developers develops the software in a Familiar and in-house environment where no tight deadline exists. For e.g., payroll, Inventory projects etc.
- **Semi- Detached Mode**: Typically 50-300 KLOC sized projects come in this mode category. A medium size team of average experience developers develops the software in medium constraints of environment and deadline. For e.g., Utility systems like compilers, Database systems, editors etc.
- **Embedded Mode**: Typically over 300KLOC sized projects come in this mode category. A team of very little previous experience developers develop the software's of Real time systems and complex interfaces where a tight deadline exists. For e.g., ATMs, Air Traffic control etc.

COCOMO II: Need is the mother of invention. This phrase proofs its truthiness in this case also. Problems with COCOMO1 lead to the formation of COCOMOII model. COCOMO1 was introduced in 1981. But in late 90's, COCOMO I faced a lot of problems in estimation of new life cycle processes such as non-sequential and rapid development process models, reuse-driven approaches, and object-oriented approaches. COCOMO II considers all these factors with some improvements in COCOMO1. COCOMO II was published initially in the Annals of Software Engineering in 1995 with three sub models; an application-composition model, an early design model and a post-architecture model.

FIVE SCALE FACTORS: A detailed study conclude that most significant input to the COCOMO II model is size, so, a good size estimate is very important for any good model estimation. Size in COCOMO is treated as a special cost driver, so it has an exponential factor, E. The exponent E in effort is an aggregation of five scale factors. All scale factors have rating levels. These rating levels are Very Low (VL), Low (L), Nominal (N), High (H), Very High (VH) and Extra High (XH). Each rating level has a weight, W, which is a quantitative value used in the COCOMO model. The five COCOMO II scale factors with their description and ratings are shown in table:

Table1: COCOMO II Scale Factors

Description	Scale factor	Very low (0.05)	Low(0.04)	Nominal(0.03)	High(0.02)	Very high(0.01)	extra high (0.00)
Precedentedness	PREC	Thoroughly Unprecedented	Largely Unprecedented	Somewhat Unprecedented	Generally Familiar	Largely Familiar	Thoroughly Familiar
Flexibility	FLEX	Rigorous	Occasional relaxation	Some relaxation	General Conformity	Some conformity	General goals
Risk Resolution	RESL	Little (20%)	Some (40%)	Often (60%)	Generally (75%)	Mostly (90%)	Full (100%)

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

Team cohesion	TEAM	Very difficult interaction	Some difficult	Basically cooperative	Largely cooperative	Highly cooperative	Seamless Interaction
Process Maturity	PMAT	Weighted average of "yes" Answers to CMM maturity Questionnaire					

CODE CATEGORY

For the determination of size, code is the basis. But code can have different sources. These sources are briefly discussed here:

- **New code**: new SLOC that is to be written. New code written manually (without any ICASE; for ICASE see Applications Composition Model) different worded: from scratch.
- **Reused code**: Code is reused as it is pre-existing code that is treated as a *black-box* and plugged into the product as it is.
- **Adapted code**: Code is reused but with some modification. Pre-existing code is treated as a *white-box* and is modified for use with the product.
- **COTS component**: code is not available as different organizations made it, but licensed to be used. Commercial off-the-shelf component, leased, licensed, source code is not available.
- **Automatically translated code**: Code is reused but with the help of tools. Pre-existing code for which translation is supported by automated tools.

LOC and FP

LOC (LINES OF CODE): This is the number of lines of the **delivered source code** of the software, excluding comments and blank lines and is commonly known as **LOC** . Although LOC is programming language dependent, it is the most widely used software size metric. However, exact LOC can only be obtained after the project has completed.

There are two methods for counting the lines of code:

- **Logical:** A logical count includes each logical statement that is terminated by a source statement delimiter such as semicolon, colon or period.
- **Physical:** A physical count includes each line of code that is terminated physically (by hitting the ENTER key of a computer keyboard, which completes the current line and moves the cursor to the next line.

Expressing software size in terms of LOC creates a lot of problems:

- No accepted standard definition exists for LOC.
- The number of LOC depends on the programming language used and the individual programming style. Hence, counting variations can also affect the size of application written in multiple languages. Thus, it is not possible to directly compare the productivity of projects developed using different languages.
- LOC are difficult to estimate early on in the project life cycle. However, it is necessary to be able to predict the size of the final product as early and accurately as possible, if cost models based on system size are to be useful.
- Finally LOC emphasizes coding effort, which is only one part of the implementation phase of a software project.

FUNCTION POINT: To overcome the problems of LOC, a measure of size is formed that capture the size of a software product in terms of its functional characteristics. Instead of counting the LOCs of a developed solution for the problem, it counts the size of the problem to solve. A Function Point was firstly proposed by Albrecht and Gaffney. The main motive of designing the Function Points is to provide a compatible environment for both software developers and users so that they can easily define their functional requirements.
Mechanism to consider this functionality is to define five basic Functions. Two of these functions addresses the data requirements of the end users and are referred to as Data Functions. The remaining three addresses the user's needs to access data and are referred to as Transactional Functions.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

The Five Components of Function Points

Data Functions

- Internal Logical Files (ILF)
- External Interface Files (EIF)

Transactional Functions

- External Inputs (EI)
- External Outputs (EO)
- External Inquiries (EI)

Weight multiplied to nos. of these components and their sum are called **Unadjusted Function Points(UFP).**

But this value is not exact as till now certain project characteristics that influence software project are not considered, there are 14 such **Technical Adjustment Factors** whose consideration makes **Unadjusted** to **Adjusted Function Points.**

Sum of these Adjustment Factors is called as **Technical Complexity Factor(TCF).**

UML and UCP

UML: Object oriented development is the combination of object oriented analysis, object oriented design, and object oriented programming. In object oriented analysis we are preparing the model which helps us in requirements gathering. In object oriented design can be done by using the UML. In object oriented programming project construction can be done.

UML notations and diagrams are as follow:

- Use Case Diagram
- Class Diagram.
- Sequence Diagram
- Interaction Diagram
- Collaboration Diagram
- State chart Diagram
- Activity Diagram

USE CASE POINT ESTIMATION TECHNIQUE: This technique was proposed by Gustav Karner in 1993 to estimate the projects based on OO. The main aspects of use case point estimate technique are actors, use cases, associations between actors and use cases, relationships between actors, relationships between use cases. 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 an actor and the target system, the event being performed entirely or not at all. The basic formula for converting all of this into a single measure, use case points, is that we will weigh the complexity of the use cases and actors and then adjust their combined weight to reflect the influence of the nonfunctional and environmental factors.

The UCP counting process consists of the following steps :-

- Unadjusted Actors Weights (UAW)
- Unadjusted Use Case Weights (UUCW)
- Unadjusted Use Case Points (UUCP)
- Technical Complexity Factor (TCF)
- Environmental Factor (EF)
- Adjusted Use Case Points (AUCP)

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

- Staff hours per Use Case Point.

Classifying Actors and Use Cases: The first step is to classify them into simple, average or complex:

Table 2: Classifying Actors

Actor Type	Weighting Factor
Simple	1
Average	2
Complex	3

- **Simple Actor:** It represents another system with defined Application Programming Interface(API).
- **Average Actor:** Its another system interacting through a protocol such as TCP/IP.
- **Complex Actor:** It maybe a person interacting through a GUI or webpage.

The total unadjusted actor weights(UAW) is calculated by how many actors there are of each kind, multiplying each total by its weighting factor and adding up the products.

Table 3: Classifying Use Cases

Use Case Type	No. of Transactions	Weighting Factor
Simple	<=3	1
Average	4 to 7	2
Complex	>=7	3

Each use case is defined as simple, average or complex depending on no. of transactions in use case description including secondary scenarios. A transaction is set of activities which is performed entirely or not at all.

TECHNICAL COMPLEXITY: The total effort to develop a system is influenced by factors beyond the collection of use cases that describe the functionality of the intended system. A distributed system will take more effort to develop than a non distributed system. The impact on use case points of the technical complexity of a project is captured by assessing the project on each of thirteen factors. Many of these factors represent the impact of a project's nonfunctional requirements on the effort to complete the project. The project is assessed and rated from 0 (irrelevant) to 5 (very important) for each of these factors.

Table 4:The weight of each factor impacting technical complexity

Factor	Description	Weight
T1	Distributed Systems	2
T2	Performance Objectives	1
T3	End User Efficiency	1
T4	Complex Processing	1
T5	Reusable Code	0.5
T6	Easy to Install	0.5
T7	Easy to Use	2
T8	Portable	1
T9	Easy to Change	1
T10	Concurrent Use	1
T11	Security	1
T12	Access for third parties	1
T13	Training Needs	1

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

In Karner's formula, the weighted assessments for these twelve individual factors are next summed into what is called the **TFactor.** The TFactor is then used to calculate the **Technical Complexity Factor, TCF**, as follows:

TCF = 0.6 + (0.01* TFactor)

ENVIRONMENTAL COMPLEXITY: Environmental factors also affect the size of a project. The motivation level of the team, their experience with the application, and other factors affect the calculation of use case points.

Table5: Environmental factors and their weights

Factor	Description	Weight
E1	Familiar with the development process	1.5
E2	Application experience	0.5
E3	Object-oriented experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time staff	-1
E8	Difficult programming language	-1

The weighted assessments for these eight individual factors are summed into what is called the **EFactor**. The EFactor is then used to calculate the **Environment Factor, EF**, as follows:

EF =1.4 + ("0.03*EFactor)

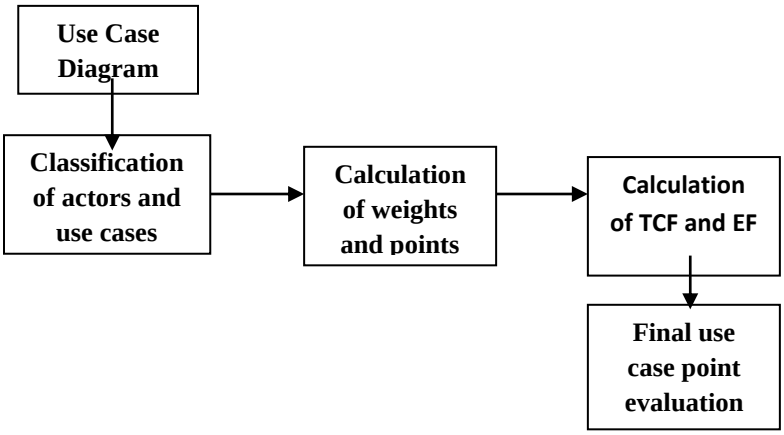


Figure 1: Steps to calculate Use Case Point

The number of use case points in a project is a function of the following:

• The number and complexity of the actors on the system.

• The number and complexity of the use cases in the system.

• Various non-functional requirements (such as portability, performance, maintainability) that are not written as use cases.

• The environment in which the project will be developed (such as the language, the team's motivation, and so on).

Advantages:

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

- The use case model is the front end model of the Unified Modeling Language (UML).With the emergence of the UML as the most commonly used notation to model and design object-oriented software products, the application of use cases for size and hence for effort and cost estimation seems to be a perfect fit.
- Reduces duration or estimation time as there are some use case tools available that automate the process of counting the number of use case points in system for e.g., U-EST.
- Reduces effort by using OOPs concept.
- Current trend issues for software requirement gathering are through modeling the use case diagrams.
- Use case textual description is written earlier within the life cycle of a system.
- Use case notations and diagrams are easy to understand and to interpret.
- Several environmental factors along with technical factor are present on the behalf of which estimates are assumed to be more accurate.
- Object-oriented paradigm and visual programming form the basis for use of UCP technique.
- Weighting factors are not much complicated.

LITERATURE SURVEY

Work done in Use Case Point Method and Effort Estimation based on use case model are given below:-

"Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department", (M. Arnold, P. Pedross, 1998): The authors proposed that the use of the use case point method is accepted to estimate the size . In this paper Arnold and other compare the object oriented methods like OOSE, OMT, OOAD, UML, ROOM, OBA, and Syntropy. Although it has been accepted by the researches that a number of syntactic and semantic definitions existed for these methods, they also conclude that no method provides the technique for size measurement based on use cases and scenario. Researchers also described that since the language concepts for documentation are not well understood, it would be important to define the language concepts more precisely in advance.

"The Estimation of Effort Based on Use Cases", (John Smith 1999): The author proposed a framework to estimate LOC from use case diagram. The framework takes account of the idea of use case level, size and complexity, for different categories of system and does not resort to fine-grain functional decomposition.

"Effort Estimation Tool Based on Use Case Points Method", (Shinji kusumoto, Fumikazu matukawa, Katsuro inoue, Shigeo hanabusa, Yuusuke maegawa): To effective introduction of UCP method, the author has developed an automatic use case measurement tool, called U-EST. This work describes the idea to automatically classify the complexity of actors and use cases from use case model. "**Cost Estimation using Extended Use Case Point (e-UCP) Model", (Kasi Periyasamy and Aditi Ghode, 2009):** This research focus on the internal details of use cases. For this a focus on the use case narratives, uses the relationships between the entities in a use case diagram, and hence closely estimates the size of the software product to be developed. The authors extended the original UCP model with additional information obtained from use case narratives also changes some parameters value.

"A Linear Use Case Based Software Cost Estimation Model", (Hasan.O. Farahneh, Ayman A. Issa, 2011): This research reports on the development of new linear use case based software cost estimation model applicable in the very early stages of software development being based on simple metric. Evaluation showed that accuracy of estimates varies between 43% and 55% of actual effort of historical test projects. These results outperformed those of well known models when applied in the same context. Further work is being carried out to improve the performance of the proposed model when considering the effect of non-functional requirements.

PROPOSED METHODOLOGY

Presently we are working in object oriented paradigm where a life seems to be automated, although manual workings are present but reduces to a very low level.

Now various researches have been performed in this context. Steps which will be followed in our research are as follows:-

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

- Since the object-oriented approach has become a *de facto* standard for software development, it became necessary to revise the effort and cost estimation models to suit the object-oriented technology.
- To reduce the efforts associated with software development.
- To reduce the cost associated with estimation of software resources.
- To use simple diagrams and notations of UML to make the estimation process simple such that even a novice developer can perform estimation accurate and reliable.
- To make the estimation process easy while working with current trend and issues of Object oriented paradigm.

RESULTS

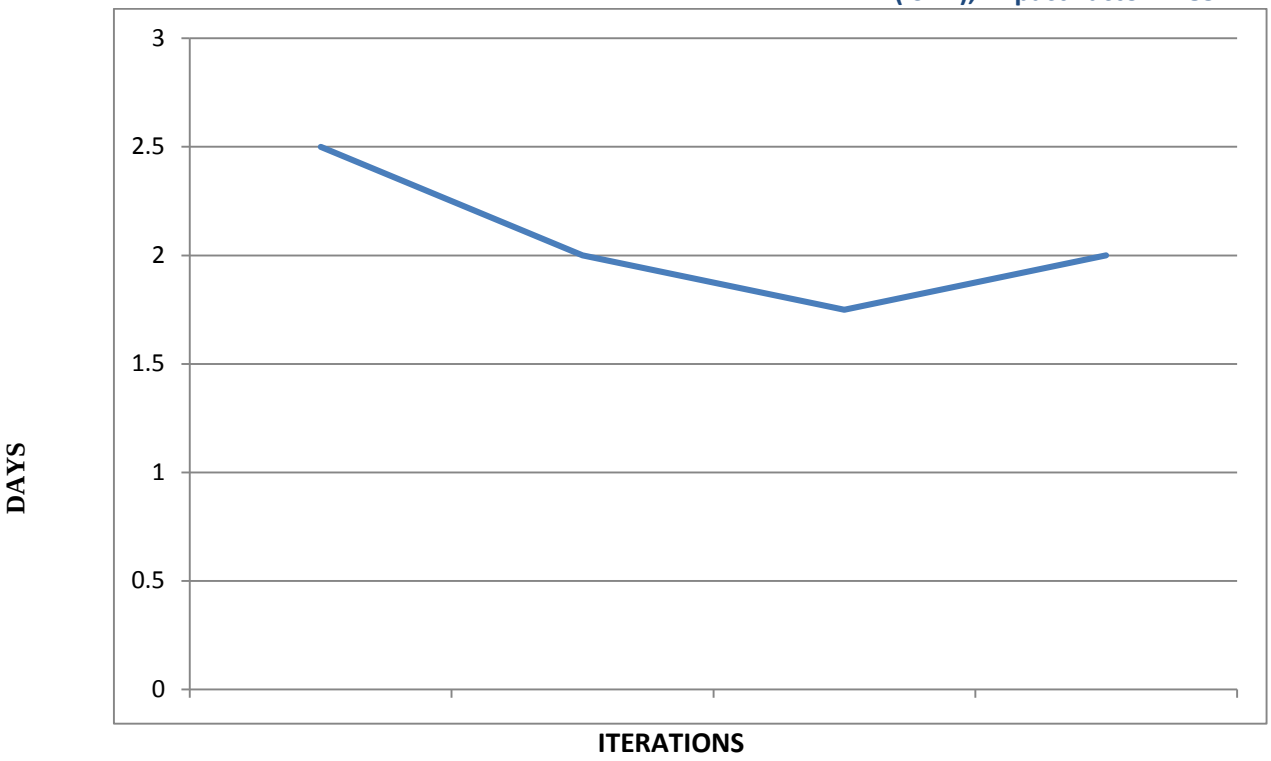
Effort estimation models COCOMO 1 and COCOMO II provides best estimates for effort and cost but for their size input parameter, developers uses FP which is not at all accurate due to the reasons specified.
So here we are describing the self implanted tool that will evaluate the outputs of Function Point, COCOMOI, and COCOMOII and Use case points (UCP) estimation technique.
To come up with our final Use Case Point (UCP) total, Karner's formula takes the Unadjusted Use Case Points (UUCP), the sum of the Unadjusted Use Case Weight (UUCW) and the Unadjusted Actor Weight (UAW) and adjusts it by the Technical Complexity Factor (TCF) and the Environmental Factor (EF). This is done with the following formula:

UCP = UUCW *TCF *EF

One of the most useful techniques to come out of agile software development is the burn down chart (Schwaber and Beedle 2001). A typical release burn down chart shows the estimated amount of time remaining in a project as of the start of each iteration. The sample burn down chart shows a project that had approximately 2.5 days of work at the start of the first iteration, about 2 by the start of the second iteration, and about 1.75 by the start of the third iteration. Things didn't go well during the third iteration, and by the start of the fourth iteration the team was back to an estimate of 2 days of work remaining. The cause of this increase is unknowable from the burn down chart. But this is usually the result of adding new requirements to the project or of discovering that some upcoming work had been incorrectly estimated**.**

Figure 2: Sample Burn down Chart

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852



ANALYSIS: It has been clearly shown from the table given below that the Use case technique reduces effort while gathering functional requirements of future system.

Table 6

	FUNCTION POINT		FUNCTION POINT		
	COCOMO1 Basic Model	COCOMO 1 Intermediate model	COCOMO II Early Design Model	COCOMO II Post-Architecture Model	UCP
Efforts	11 PM	12 PM	21 PM	11PM	**10PM**
Duration	6 M	6M	9M	8M	1457 H
Cost	111200	121200	201200	111200	**101200**

Where efforts are in PM. PM Stands for Person Months

And the Cost is measured in Rupees, where a standard fixed cost is assigned in all the techniques.

Hence the result shows that there is reduction in efforts and cost while preserving the trend of Object Orientation in scope, and the objective of the thesis is being met.

CONCLUSION

The tool will help in requirement gathering in object oriented paradigm. The result shows that effort and duration is less comparative to the COCOMO 1and COCOMO II. The cost will reduce accordingly as efforts.

Object-oriented design and development are popular concepts in today's software development environment.

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

In conclusion, use case points method of effort estimation is a very valuable addition to the tools available for the project manager. The method can be very reliable or just as reliable as other effort estimation tools such as COCOMO, function point and lines of code. All of the estimation methods are susceptible to error, and require accurate historical figures for productivity in order to be useful within the context of the organization. Use case points method is especially valuable in those system development projects where use cases are produced anyway. It is comparable to the function point method that has become quite respected throughout the industry. It provides estimates that are sometimes better than what experts can provide to the industry. Expert estimates should not be excluded from the process of estimation; rather it should be used in conjunction with use case estimates to ensure an accurate estimate. Lastly, with standardization and kind of national and international efforts that have helped the function point method become widely accepted, this method also has the potential to become a mature and widely accepted estimation tool.

REFERENCES

[1] Gautam Banerjee, *"*Use Case Points- An Estimation Approach*"* Aug. 2001.
[2] Mel Damodaran, **"**Estimations using Use Case Points"
[3] Encyclopedia of Software engineering, volume 2, second edition
[4] Nancy Merlo – Schett, **"**Seminar on Software Cost Estimation*",* Computer Science, 2002
[5] Rajiv aggarwal book. K k aggarwal.
[6] Bharat Bhushan Aggarwal
[7] Hareton Leung Zhang Fan, "Software cost estimation"**,** Department of Computing, The Hong Kong Polytechnic University, pg no.2-8.
[8] B.W. Boehm et al *"*The COCOMO 2.0 Software Cost Estimation Model*"*, American Programmer, pp.2-17, July 1996.
[9] Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, *"*Cost Models for Future Software Life Cycle Processes: COCOMO 2.0*"* Science Publishers, 1995.
[10] Costar tool table.
[11] COCOMO MODEL