iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** VIII **Month of publication:** August 2017

DOI: http://doi.org/10.22214/ijraset.2017.8208

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VIII, August 2017- Available at www.ijraset.com

Comparative Analysis of Software Size Estimation Techniques in Project Management

Yogesh Kumar[1], Neeraj Varshney [2]
[1]*UIET, CSE Department, Maharishi Dayanand University, Rohtak.*
[2]*B.Tech Student, BITS Pilani.*

Abstract: Estimation is an important part of the project management which influences the success of project. Inaccurate and unreliable estimation can result into failure of project. Estimation is very important part of management activity that should be done carefully. Software estimation is the process of predicting the effort & cost required to develop software. For estimation of effort, we require an accurate estimate of size. In short, all estimation techniques based on an accurate and precise size estimation which in turn used as an input for effort and cost estimation. This review paper provides a general overview of software size estimation models and techniques. Techniques for size estimation are LOC based, FP (Function Point), UCP (Use Case Point), OP (Object Point). There is no single technique that can be termed as the best technique but a careful comparison of the results of several approaches is most likely to produce realistic estimates.
Keywords: Software size estimation, FP, UCP, LOC, OP and Project management.

I. INTRODUCTION

Project management is one of the most important activities performed throughout the software projects. Main phases of project including analysis, design, implementation and deployment are entirely dependent on project management process. All policies, milestones and responsibilities are organized in project management plan. It is undeniable that planning and scheduling of project is a critical part of project management regardless of project type. In first steps of project, project management team should decide on several important questions related to project planning such as how to arrange development team, how to distribute the responsibilities, how to determine deadline for artifacts, how to determine the duration of project and so on. Appropriate response to these questions can ensure the success of software project. On the other hand, careless answering and lack of attention to planning aspects of project may lead to project fault. Knowledge of project management team regarding the project attributes has a considerable effect on dealing with the mentioned questions.[2]

One of the most critical activities in software project management during the project inception phase is to estimate the effort and cost needed to complete the project tasks. Decisions related to Financial and Human resource aspects are dependent upon the estimations of the effort needed to make software. The common problems with estimation are either related to the over estimation or the underestimation of the effort needed. The consequences of underestimation lead to low employee morale, decline in reputation and a stressful work environment. On the other hand, over estimation can lead to too many resources committed to the project or losing the bid or making poor decisions related to outsourcing parts of the project versus developing it internally. In general, the tendency in the software industry is to underestimate the effort and that leadsto the realization during the project execution that the milestones cannot be met.[7]

In recent years, software has become the most expensive component of Computer system based projects and therefore the role of proper estimation in a software project is important. Many estimation models have been proposed over the last 40 years. Although most of these researchers started working on developing models of estimation at about the same time, they all faced the same dilemma: as software grew in size and importance it also grew in complexity, making it very difficult to accurately predict the cost/effort of software development. Just like in any other field, the field of software engineering estimation models has its own pitfalls. The fast changing nature of software development has made it very difficult to develop parametric models that yield high accuracy for software development in all domains.[7].

II. DESCRIPTION OF ESTIMATION

Management in any project starts with estimation. An effective estimation is the back bone for the development of any project. Without effective estimates proper project planning and tracking is impossible. If the estimates are too low then the project

management tries to employ more personnel in order to expedite the development process; that eventually results in poor quality product and employee dissatisfaction. [2]

Basic software estimations are as follows:

Estimation of the size.

Estimation of the effort.[18]

Estimation of the schedule.

Estimation of the cost .[18]

In this review paper, the techniques for estimation of size of projects are explained and their comparison is presented.

III. SIZE ESTIMATION

A sound size estimate could be a good foundation for the software estimation. The information source for estimation can be the project proposal, system specification or software requirement specification. If the size estimation is being done in the later stages such as design or during coding, then design specifications and other work products can be used as information source for estimation .[2]

The ways in which estimation can be done are as follows:

A. LOC based

LOC is the simplest among all metrics available to estimate project size. This method measures the size of a project by counting the number of source instructions in the developed program. While counting the number of source instructions, lines used for commenting the code and the header lines are ignored.

Determining the LOC count at the end of the project is very simple but accurate estimation of LOC at the start of the project is very difficult. Accurate estimation of size acts as a base for other estimation of project activities and any wrong estimation lead to wrong estimates of other dependent activities. In order to estimate the LOC count in the beginning of a project, we have to make a systematic guess. Every project cannot be taken as a single problem, it is divided into several sub problems and further in modules by project managers until the size of deepest level module can be predicted. To make our estimation precise we can use the past experience of similar projects. After predicting and estimating lowest level module size, the total project size can be estimated by adding estimates level by level.[3]

VERSIONS of LOC[4]

1) *DSI (Delivered source instruction)*
2) *KDSI (kilo (Thousands of) delivered source instruction)*

B. Function Point Analysis (FPA)

As the system grows in size, it is really hard to estimate the size of the software early in the development. Divide and conquer has been the best strategy for tackling bigger problem for decades. Function point analysis, introduced by Allan J Albrecht of IBM in late 1970s, follows the concept of divide and conquer strategy for estimating the size of any software. [2]

FPA breaks the system into smaller pieces so that intricacies of the systems become more visible and can be analyzed better. Function point analysis measures size of the software on the basis of the functionalities to be provided by the software. The method quantifies the functionalities of software by the information provided by the user based on logical design. FPA estimates the size of software in terms of function point counts (FPC) which can be converted into SLOC easily if the equivalent SLOC for unit FPC is available.

Size estimation is critical issue for all kind of projects i.e. for development, enhancement and application projects. On the basis of the categories of the projects function points can also be categorized into following categories [2]:

1) *Development Project Function Point Count:* When the project is under development, the amount of information about the project varies from phase to phase. Development Project Function Point Count is useful for the projects under development. It can be used in any phase. Using FPA in every phase of development allows tracking of size overrun.
2) *Enhancement Project Function Point Count:* Every software goes through the enhancement stage either addition of new functional requirement or a non-functional requirement. Enhancement project function point count tries to estimate the size of enhancement projects. It helps in understanding the movement of a project from development stage to enhancement stage.
3) *Application Project Function Point Count:* Once the application is developed function points can be calculated to make baseline for future uses. It can be used for predicting maintenance size.

Complexity of software and the effort needed to develop it are a function of the number and type of five different kinds of functional components that can be obtained and assessed at the requirements specifications phase [5].

4) *Internal Files (IF):* corresponding to the database files that are created and maintained within the application to develop.
5) *External Files (EF):* corresponding to the files owned and maintained by other applications, but are used by the application to develop.
6) *External Inputs (EI):* corresponding to the inputs that affect the control flow and internal logic of the application leading to the creationand maintenance of data.
7) *External Outputs (EO):* corresponding to the data leaving the application to different output devices, files or external systems.
8) *External Inquiries (EIQ):* corresponding to simple user queries resulting in responses to them.

The first two types of components are referred to as data-based components and the other three as transaction-based components. Each component is then assigned a points value on the basis of its type and complexity. The points values of all the components are then summed (see equation 2.1) to give a size for the system in unadjusted function points (UFPs).

$$UFPs = \sum(EI) + \sum(EO) + \sum(EF) + \sum(EIQ) + \sum(IF) \quad \ldots(2.1)$$

The technical complexity factor is given by quantifying the effect of fourteen General Application Characteristics that affect the complexity of carrying out the design and implementation task.

The General Application Characteristics are shown as.

TABLE I :
WEIGHT FACTORS FOR FUNCTION POINT METRICS

Function Type	**Simple**	**Average**	**Complex**
External Input (EI)	3	4	6
External Output (EQ)	4	5	7
External Inquiries (EIQ)	3	4	6
External Files (EF)	7	10	15
Internal Logical Files (ILF)	5	7	10

TABLE IIIII :
GENERAL APPLICATION CHARACTERISTICS AFFECTING THE COMPLEXITY OF SOFTWARE PROJECTS

1. Reliable Backup & Recovery
2. Data Communication
3. Distributed Functions
4. Performance
5. Heavily Used Configuration
6. Real- Time Data Entry
7. Ease of Use
8. Real-Time Updates Needed
9. Complexity of the Interface
10. Complexity of the Processing
11. Reusability
12. Ease of Installation
13. Multiple Sites
14. Easy to Change

The degree of influence of each of the characteristics can range from zero (meaning, not present, or has no effect) to five (meaning, a strong influence throughout). The sum of the fourteen characteristics, that is, the total degrees of influence (DI), (given in equation 2.2) is then converted to the technical complexity factor (TCF) using the formula given as (equation 2.3).[7]

$$DI = \sum \text{General Application Characteristics [i]} \quad \text{where i= 1 to 14....(2.2)}$$

$$TCF = (0.65 + 0.01 * DI) \quad \ldots(2.3)$$

The TCF is now used to modify the size of the system to give the overall size in function points as below.

FPs = (UFP * TCF) ….(2.4)

TABLE III : TECHNICAL COMPLEXITY FACTORS FOR UCP METRICS

Technical Complexity Factors for UCP Metrics	
Technical factors	**Weight**
Distributed Systems	2
Performance Requirement	1
End-User Efficiency	1
Internal Processing	1
Reusability of Code	1
Installation Ease	0.5
Usability Requirement	0.5
Portability Requirement	2
Changeability Requirement	1
Concurrency	1
Security Requirement	1
Direct Access to Third Party	1
User Training facility	1

The mappings between FP to LOC for different languages are decided by International Function Point User Group (IFPUG). Another estimation technique namely "***Feature point***" extends the function points [6].

C. Use Case Points (UCP)

The Use Case Point (UCP) is a software effort estimation technique that was introduced by Kamer in 1993. UCP is an extension of the function point method based on the use cases existing in the use case model of a software system. The unadjusted actor weight is the sum of complexity values assigned to each actor. Similarly, the unadjusted use case weight (UUCW) is the sum of complexity values assigned to each use case. The total unadjusted use case point (UUCP) is the sum of UAW and UUCW. The number of adjusted use case points (AUCP) is computed by multiplying the UUCP with the product of two adjustment factors: the technical complexity factor (TCF) and the environmental factor (EF).[7]

TABLE IVV:ACTOR COMPLEXITY VALUES

Actor Complexity Values	
Actor Type	**Value**
Simple	1
Average	2
Complex	3

UAW = ∑(Actor Complexity Values) ....(3.1)
UUCW= ∑ (Use-Case Complexity Values)…(3.2)

TABLE V:USE-CASE COMPLEXITY VALUES

Use-Case Complexity values	
Use-Case Type	Value
Simple:≤ 3 Transactions	5
Average: Between 4 and 7 Transactions	10
Complex: ≥ 7 Transactions	15

TCF= 0.6 + 0.01 x ∑(Technical Complexity Factors [i]) where i=1 to 13 …...(3.3)

TABLE VV:ENVIRONMENTAL FACTORS

Environmental Factors for UCP Metrics	
Environmental factor	Weight
Familiarity with Project	1.5
Application Experience	0.5
Object-Oriented Experience	1
Lead-Analyst Capabilities	0.5
Motivation	1
Stability of Requirements	2
Part Time staff	-1
Programming Language Difficulty	-1

$$EF = 1.4 - 0.03 \times \sum(\text{Environmental Factors}[i]) \text{ where } i=1 \text{ to } 8 \quad \text{…..(3.4)}$$

$$AUCP = (UAW + UUCW) \times TCF \times EF \quad \text{…..(3.5)}$$

There are many uses of Function Points beyond estimating schedule, effort and cost as discussed in preceding sections. Many organizations are using function points and software metrics just to report organizational level trends. Many project teams report data to a central metrics group and never see the data again. It is equivalent to reporting data into a black-hole. If project managers begin to understand how Function Points can be used to estimate costs, productivity, quality, test cases, to calculate maintenance costs, and so on, they will be more likely to invest in counting Function Points, making an effective use of FP.[14]

D. Object Points (OP)

It measures the size from a different dimension. This measurement is based on the number and complexity of the following objects: screens, reports and 3rd Generation Language (3GL) components [8], [9], [17]. This is a relatively new measurement and it has not been very popular. But because it is easy to use at the early phase of the development cycle and also measures software size reasonably well. This measurement has been used in COCOMO II for cost estimation. Following are the steps for calculating object point:

1) Assess object count: number of screens, reports and 3GL components.

2) Classify object: simple, medium and difficult depending on the values of characteristic dimensions.

3) Weight the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level as given in the following table.

TABLE VVIVII :WEIGHT VALUE BY OBJECT COMPLEXITY

Object Type	**Simple**	**Medium**	**Difficult**
Screen	1	2	3
Reports	2	5	7
3 GL Components	10	10	10

TABLE VVIIIIXX :PRODUCTIVITY

	Very Low	**Low**	**High**	**Highest**
PROD	4	7	25	50

4) Determine object points: add all the weighted object instances to get one number, the object point count.

5) Estimate percentage of reuse you expect to be achieved in this project. Compute new object points to be developed as,

$$NOP = (\text{Object Point}) * (100 - \%\text{reuse})/100$$

where,

%reuse is the percentage of screens, reports, and 3GL modules reused from previous applications.

6) Compute the estimated person-months as, Person Month = NOP/PROD.

Object points are easier to estimate from a specification than function points as they are simply concerned with screens, reports and 3GL modules. They can therefore be estimated at an early point in the development process. At this stage, it is very difficult to estimate the number of lines of code in a system.

IV. COMPARISON OF VARIOUS SIZE ESTIMATION TECHNIQUES

TABLE IX

Technique	Strength	Weakness
1.LOC(Line of Code)	• Simple measurement. • Programs can be developed to automate the process of counting LOC. • Physical entity so effects can be visualized.	• Biggest problem: inaccurate. • It is only for code. It cannot measure other attributes like specification. • Its scope is restricted to coding part only • Language dependent • Lack of cohesion and functionality • Dependent on experience of developer • More irrelevant in context of GUI based languages/tools. • Far from OO development • Lack of counting standards.
2.FPA (Function Point Analysis)	• Measures system from functional perspective.[19] • Doesn't depends upon language, development method, h/w or platform etc.[19] • Wide Scope: requirement, analysis, code, design, testing and deployment. • Helps in judgment of requirement completeness and in early stages of development.[19] • Data from past projects can be used to improve the accuracy. • Availability of empirical formula. • Better benchmarking. • Better than LOC	• Counting can't be automated • Requires significant level of details • Ignores quality of output • Requires experience • Not suitable for maintenance work. [2]

3.UCP(Use Case Points)	• Simple. • Fast. • Can be automated • Can be enhanced using fuzzy logic, neural network methods. • Duration can be derived from use case points.[12] • Pure measure of size.[12]	• Absence of graduation when classifying the complexity of use case.[11] • Doesn't distinguish between large, very large and super large use case.[11] • No standard method for conversion between size in UCP and in FP or LOC. • Some of technical factors do not have impact across overall project.[12]
4.OP(Object Points)	• Suite of multi-dimensional size metric.[13] • Easy to count and to learn to count.[13] • More correlated with effort than FP.[13] • Extensible and modifiable size metric.[13]	• New and not very popular • Knowledge of application domain is essential. • In large project, more time required for team communication. • Not clear how productivity and quality metrics are related.

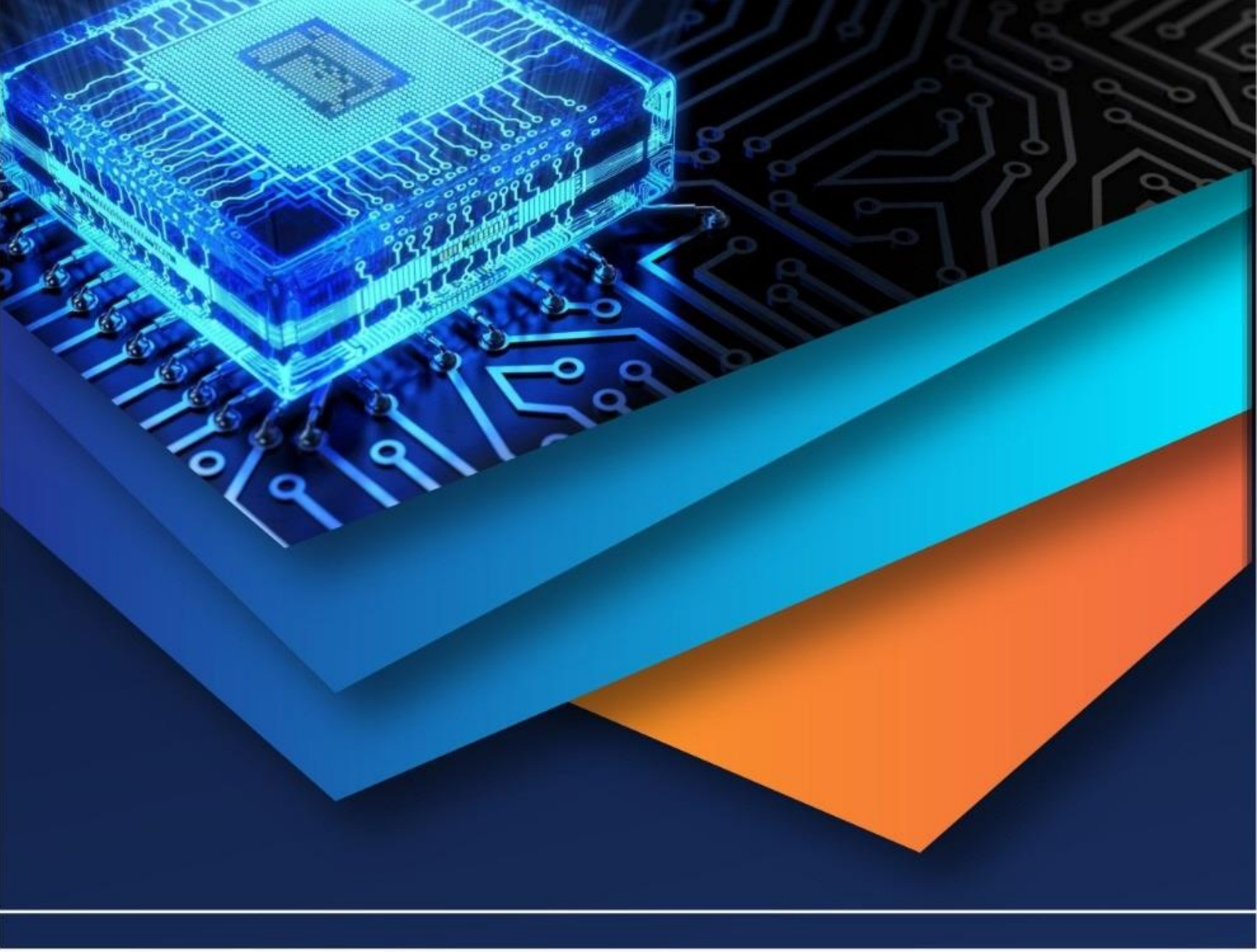
V. CONCLUSION

From the discussion of above techniques we can say that every method have advantages and disadvantages. In any software project, accurate estimation cannot be generated by using just one technique .in order to make precise estimation, combination of techniques and data from past projects should be considered. It is highly recommended that organizations employ Function Points, develop expertise, elicit accurate data, build a good repository of historic projects, and in turn use the data for effective benchmarking and continuous improvement [14].

REFERENCES

[1] Kathleen Peters, Berkun, Scott, "*Art of Project Management. Cambridge"*, MA:O'Reilly Media. ISBN 0-596-00786-8. Software Project Estimation (White paper), Software Productivity Centre Inc. (SPC) in Vancouver, British Columbia, Canada, 2005.
[2] Nilesh Chandra Shukla, " *A Tool for Software Project Management for Estimation, Planning & Tracking and Calibration*".IIIT-Allahabad.
[3] Rajib mall, "*Fundamentals of Software Engineering*" ,PHI learning pvt.Ltd.2009 edition.
[4] Nasib Singh Gill, " *Software Engineering, Software Reliability, Testing and Quality Assurance*", Khanna Book Pubishing Edition.
[5] D. R. Jeffery, G. C. Low, and M. Barnes, "*A comparison of function point counting techniques*", IEEE Trans on Soft. Eng., vol. 19, no. 5, 1993, pp. 529-532.
[6] D. St-Pierre, M Maya, A. Abran, J. Desharnais and P. Bourque, "*Full Function Points: Counting Practice Manual*", Technical Report 1997-04, University of Quebec at Montreal,1997.
[7] S.K.Mohanty & A.K.Bishnoi, "Software *Effort Estimation Approaches- A Review*", International Journal of Internet Computing ISSN No: 2231 – 6965, VOL-1, ISS- 3 2012.
[8] Hareton Leung, Zhang Fan, "*Software Cost Estimation*", Article 2001.
[9] Jyoti G Borade & Vikas R Khalkar, "*Software Project effort and cost Estimation Techniques*", *International Journal of Advanced Research in Computer Science and Software Engineering* ISSN-2277-128X, Vol. 3, Issue 8, aug 2013.
[10] Matthias Kerstner, "*Software Test Effort estimation Methods*", 2 February 2011.
[11] Ali BouNassif, Luiz Fernando Capretz and Danny Ho, " *Enhancing Use Case Point Estimation Method Using Soft Computing Techniques*", ISSN-2229-371X.
[12] Mike Cohn, "*Estimating With Use Case Points*", Founder-Mountain Goat Software.
[13] Erik Stensrud, "*Estimating with Enhanced Object Points vs. Function Points*". Andersen Consulting Drammensveien 165, 0212 Oslo, Norway& University of Oslo, Dept. of Informatics
[14] Kurmanadham V.V.G.B. Gollapudi, "*Function Points or Lines of Code? – An Insight*", Wipro Technologies.
[15] Hihn, J., Gemoetz, L., Mahmood, M. & Longstreet, D. (1997), "*The Impact of Learning on Function Point Counting Accuracy*", Presentation on 12* COCOMO Forum**,** Los Angeles, October 6-10.
[16] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, "*Object-Oriented Software Engineering: A use Case Driven Approach"*. Addison Wesley, 1992.
[17] Bogdan Stepien, "*Software Development Cost Estimation Methods and Research Trends*", Computer Science, Vol.5,2003.

[18] Chetan Nagar, Anurag Dixit, "*Software efforts and cost estimation with systematic approach*", ISSN:2079-8407, Vol.2 No.7, July 2011.
[19] Pressman, Roger S., "*Software Engineering: A Practioner's Approach*", 6th Edn., McGraw-Hill New York, USA., ISBN: 13: 9780073019338,2005
[20] Sean Furey, "*Why We Should Use Function Points*", IEEE Software, March/April 1997, Vol. 14 Issue 2, p28



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY