IJCTA, 9(22), 2016, pp. 75-83

75-83

A Survey Repo rious Software Estimation Tech nd Practices

Deepak Nandal* Dr. Om Prakash Sangwan**

Abstract : Paper offers thoroughly examine of software and project analysis methods established in industry and literature, its skills and flaws The Software Estimation is very important task for completing the project successfully. A successful software project development not only relies on the product efficiency but also the accurate estimation The estimation in software development depends on various factors especially on cost and effort factors for which further AI(Artificial Intelligence) and Algorithmic models have been put into usage. The low accuracy and non- reliable structures of the algorithmic models led to high risks of software projects. So, it is needed to estimate the cost of the project annually and compare it to the other techniques. The Metaheuristic algorithms have been developed well lately in software fields. Metaheuristics like Genetic Algorithms (GA) and Ant Colony Optimization (ACO) solve the problems according to the optimization of the problems and are very efficient in optimizing the algorithmic models and the effective factors in cost estimation. This review aims to discuss the methods for calculating and optimizing the metrices by using metaheuristic algorithms for software development. For this purpose survey of already implemented meta- heuristic algorithms like MOPSO, Bee Colony Optimization and Firefly is done.

Keywords : Software Quality, Effort Estimation, Metaheuristic Optimization.

1. INTRODUCTION

Software estimation has constantly become an dynamic research field. Exact software estimation is appealing in any software strategy, not just to perfectly schedule budget, resources, time and cost and eliminate overflow but also to sensibly estimate as software companies with improved estimates and planning will probably be able to get the projects in bidding process. Pre- bid estimation is important in acquiring endeavor for the organization. Precision of pre-bid estimation controls the smooth operating and success of a strategy. Estimation production creates the foundation for following project plan and activities as well as client dedications. According to F.Shull [1] efficient feature is achieved through innovation, cost management and quick reaction for customers. Consequently, software process change is inescapable and must preferably be based on software measurement programs [2]. However, dilemma is: synthesized traditional data essential for software, project estimation and measurement programs is not available, available but erroneous or not utilized for software size, effort estimation. Time lines are insufficient for the estimation exercise, resources are not skilled, well- equipped etc. Processes must to be aided to smooth the overall estimation process. Practice advisable for software estimation is [3] [5][8].

Software Size computation :

1. Effort estimation in person-hour is derived from software size.
2. Cost & budget calculation.
3. Proper scheduling, resource allocation is done as a final step.

* PhD Scholar(CSE), GJUS&T, Hisar, India
** Associate Professor, GJUS&T, Hisar, India

Fig. 1[5] is a good illustration of basic project estimation procedure.

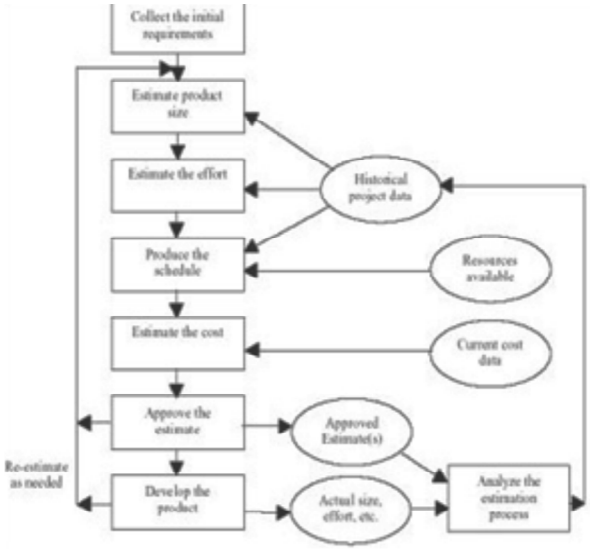

Fig. 1. Basic Project Estimation Process.

The next section presents background covering literature review and industry practices in the software estimation and project planning. Section 3 elaborates recommendations for better estimation. Section 4 summarizes the conclusion, proposed work & future direction .

2. BACKGROUND

2.1. Estimation Methodologies

Estimation methodologies [6] can be

2.1.1. Analogy Method.

In analogy approach the project to be estimated is compared with the already completed projects of that type if exists. The historical data of previously completed projects helps in the estimation. However it works only when previous data is available. Needs systematically maintained database.

2.1.2. Top Down Method.

Top down approach requires less functional and non-functional requirements and is concerned with the overall characteristics of the system to be developed. This estimation is quite abstract at the start and accuracy improves step by step. It can underestimate the cost of solving difficult low-level technical components. However top down approach takes into account integration, configuration management and documentation costs.

2.1.3. Bottom Up Method.

This method does estimation of each and every individual component and combines all components to give the overall, complete estimation of project. This approach can be an accurate method if the system has been designed in detail. However bottom up method can underestimate the cost of system level activities such as integration and documentation.

2.2. Estimation Techniques

Various techniques exist to cater the estimation procedure. These can be categorized into [3]-[4]

1. Parametric Approach like FP (Function Point), LOC etc and various model based estimations.
2. Heuristic Approach that covers Expertise Based, Learning Oriented estimations etc. All of the heuristic techniques are “soft” in that no modelbased estimation is used. There are many techniques that come under parametric as well as heuristic approaches. Few are elaborated.

2.2.1. Parametric Approaches

(*a*) LOC : Direct software size can be measured in terms of LOC (Lines of code), one of the oldest techniques. This measure was first proposed when programs were typed on cards with one line per card. Its disadvantage is that accuracy of LOC is highly dependent on the software completion and before that only expert judgment estimates can be given. Also LOC is language dependent. Stinson [20] elaborates considerations for counting LOC and Object-Oriented Software Measures.

(*b*) Function Points Metrics : Albrecht of IBM developed the Function Point metrics in 1979 [7]. In 1984, the nonprofit organization, International Function Point User Group's IFPUG [IX] set standards of Function Point Analysis and promoted the metrics and its development. British government adopted a modified form of Function Points in 1987. After that IFPUG has worked to further promote function point as an international standard on software measurement and estimation [8][22]. Some major function point metrics and extensions are discussed.

(*c*) Now days IFPUG's FPA and Charles Symons' Mark II Function Points (Mark II) are two most widely used methods for obtaining function points, a functional size measure. Function Point can be computed for a new system to be developed called [9] Development Function Points as well as for enhancement work called Enhancement Function Points.

1. **IFPUG's FPA :** In FPA [10] an estimated count is taken against Number of external inputs, outputs, Number of external inquiries, interface files, Number of internal logical files. For each domain value a low, medium or high weight is chosen. Besides the above mentioned domain values, fourteen complexity factors like Back up and recovery, Data Communication etc are given certain values as per software requirement and final estimate is calculated. Function points are simple to understand, easy to count, require little effort and practice[27]. It is independent of the technology, methodology used. Function Point is mostly used than LOC and at times more accurate than LOC, however it is abstract, difficult to automate and not a direct software size measure rather related to the functionality of a system. FP is very subjective. They depend on the estimator. FPA does not assign due importance to processing complexity. None of the FP or LOC is an ideal metric for all types of projects. FP is suitable for MIS applications.
2. **Feature Points :** To apply FP logic to software such as system software, communication software etc, there is another method called Feature Points. Feature point extends the function points [9] to include algorithms as a new class [11]. It has an additional parameter to measure algorithms with a weight of 3. Feature point model is not in popular use however this measurement is especially useful for systems with few input/output and high algorithmic complexity, such as mathematical software, discrete simulations, and military applications.
3. **3-D Function Points :** 3-D function points [12] focuses on the problem of applying function points to scientific and real-time applications. The model uses three dimensions to measure data, function and control. It has also been claimed that the model is suitable for projectoriented development.
4. **UKSMA Mark II FPA :** Symons [13]-[14] proposed a new variant based on Albrecht's FPA method in 1988. He concluded that besides 14 influential factors cited by Albrecht 6 more factors would be helpful. This metric was called Mark II FPA. UKSMA (United Kingdom Software Metrics Association) maintains Mark II FPA standard.
5. **Cosmic Full Function Points :** Cosmic-FFP [I], [II], [III] and FP-II is another improved form of IFPUG standard. Cosmic FFP (Cosmic Full Function Points) has been quite successful in NASA software estimation. The COSMIC-FFP method was first published in late 1999 [16] after proposed by St-Pierre et al. [15] and became stable with the publication of an International Standard definition in 2003. COSMIC-FP is now a standard (ISO/IEC 19761:2003)

Three other methods, IFPUG originating in the USA, Mark II FPA method from the UK and the NESMA method from the Netherlands have also been approved by ISO. These three first generation functional sizing methods were all designed 10 – 20 years ago to work for business application software. Compared with 1st Generation Measurement methods, the COSMIC-FFP method has the following main advantages and benefits.

1. The COSMIC-FFP method can be applied to business, real-time and infrastructure Software *e.g*. operating system software, telecom and process control) And to hybrids of all these types.
2. COSMIC-FFP underlying concepts are compatible with modern methods of determining software requirements and constructing software, the method is easier to learn and apply, and automatic sizing is possible with the right tools.

(*d*) COCOMO and COCOMO-II. Constructive Cost Model (COCOMO) was first proposed by Barry W. Boehm [19]-[20]. An empirical well-documented, independent model not tied to a specific software vendor, based on project experience is quite popular for software cost and effort estimation. The most fundamental calculation in the COCOMO model is the use of Effort Equation to estimate the number of Person-Months required to develop a project.

$$\text{Effort} = A \times (\text{Size})^{B}$$

Where A is proportionality constant and B represents economy. B depends on the development mode. The estimate of a project's size is in SLOC.

To get the respective results COCOMO takes LOC. COCOMO- II takes LOC, Function or Use Case points as software size input. COCOMO model [21]-[22] is provided for three operational modes:

1. **Organic.** Applied in projects that have a small, experienced development team developing applications in a familiar environment.
2. **Semi-detached.** Semi-detached mode is for projects somewhere in between.
3. **Embedded.** Embedded mode should be applied to large projects, especially when the project is unfamiliar or there are severe time constraints.

COCOMO development mode comes in three models

1. Basic
2. Intermediate
3. Detailed

Each providing progressively more accurate estimates

1. Basic COCOMO Model.

A brief overview of basic COCOMO is illustrated.

Mode	*Effort (Cost)*	*Schedule*
Organic	$\text{Effort} = 2.4(\text{Size})^{1.05}$	$\text{Time} = 2.5(\text{Effort})^{0.38}$
Semi-detached	$\text{Effort} = 3.0(\text{Size})^{1.12}$	$\text{Time} = 2.5(\text{Effort})^{0.35}$
Embedded	$\text{Effort} = 3.6(\text{Size})^{1.20}$	$\text{Time} = 2.5(\text{Effort})^{0.32}$

Figure 2. Equations for basic COCOMO Model Effort is presented in person months, size is estimated in KSLOC, and the time is estimated in months.

For example Organic mode project, 32KLOC

1. PM = 2.4 (32) 1.05 = 91 person months
2. TDEV = 2.5 (91) 0.38 = 14 months
3. N = 91/15 = 6.5 people (Personnel requirement: N = PM/TDEV)

Some other models from COCOMO suite are :

1. **COCOTS :** Cost estimation model to capture explicitly the most important costs associated with COTS-based software development and maintenance. The model is still in the experimental stage and being evolved [VI].
2. **COQUALMO :** COQUALMO [IV] (Constructive Quality Model) formerly called CODEFMO is an estimation model that can be used for predicting number of residual defects/KSLOC (Thousands of Source Lines of Code) or defects/FP (Function Point) in a software product. It can be applied in the early activities such as analysis and design as well as in the later stages for refining the estimate when more information is available. The model is based on expert-judgment.
3. **CORADMO :** This model [VII] is an extension of COCOMO for projects developed using RAD (Rapid Application Development) techniques.
4. **COPSEMO :** The intent of the Constructive Phased Schedule and Effort Model [VIII] is to calculate/ predict the schedule (months, M), personnel (P), and adjusted effort (person-months, PM) based on the distribution of effort and schedule to the various stages. COPSEMO is a part of CORADMO. Currently, a Microsoft Excel implementation of COPSEMO has been developed
5. **COPROMO :** The Constructive Productivity Model focuses [V] on predicting the most cost effective allocation of investment resources in new technologies intended to improve productivity.

(*e*) Object Points : Object points (alternatively named as application points) are an alternative function-related measure to function points when 4GLs or similar languages are used for development. Object points are easier to estimate from a specification than function point, as they are simply concerned with screens, reports and programming language modules. They can therefore be estimated at a fairly early point in the development process. At this stage, it is very difficult to estimate the number of lines of code in a system. However it cannot be used for languages prior to 4GLs.

(*f*) Putnam's SLIM : Putnam's SLIM [4], [23] (Software LIfe Cycle Management) is an automated 'macro estimation model' for software estimation based on the Norden / Rayleigh function. SLIM uses linear programming, statistical simulation, program evaluation and review techniques to derive a software cost estimate. SLIM enables a software cost estimator to perform the following functions

1. Calibration
2. Build an information model of the software system.
3. Software sizing: SLIM uses an automated version of the lines of code (LOC) costing technique. The central part of Putnam's model is called software equation as follows:

$$S = E \times (\text{Effort})^{1/3} t_d^{4/3}$$

Where *td* is the software delivery time; E is the environment factor that reflects the development capability, which can be derived from historical data using the software equation. The value of the environmental factor can vary from as little as 610 up to 57314. Putnam's recommended figures for different types of projects are Real-Time Embedded 1500, Batch Development 4894, Supported and Organized 10040. The size S is in LOC and the Effort is in person- year. Another important relation found by Putnam is

$$\text{Effort} = D_0 \times t_d^3$$

Where D_0 is a parameter called manpower build-up, which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software).

In order to use the SLIM model the software size must be identified in advance. Assessment of SLIM tells that SLIM estimates are extremely sensitive to the technology factor. It is not much suitable for small projects. It uses linear programming to consider development constraints on both cost and effort can be considered as an advantage.

(*g*) Use Case Estimation : Now-a-days estimation with use case [10] is also gaining popularity as most of the functional and non-functional requirement is captured initially in the form of Use cases. However because of many different styles and formats of use cases, this technique is a little problematic. Also just a count of use case, scenarios per use case didn't give an accurate idea of the complexity of problem to be solved. Still there are many case studies and papers that revealed the success and effectiveness of use case estimation.

For parametric computations, three estimates optimistic, likely and pessimistic are computed and an average is considered.

2.2.2. Metaheuristics Approach

A metaheuristic is a high-level problem-independent algorithmic frame-work that provides a set of guidelines or strategies to develop heuristic optimization algorithms. But a concrete definition has been elusive and in practice many researchers and practitioners interchange these terms. Thus, the term metaheuristic is also used to refer to a problem specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework[10][15].

Metaheuristic [17][18] are master strategies for the solution of problem under some conditions for instance MODS and Ant Colony inspired algorithms. It is a semi- mythical method for finding good heuristics for particular problems. For example: "What parameter settings do I use to get good results when applying heuristic method X to problem Y?" Meta heuristics are general-purpose algorithms that can be applied to solve almost any optimization problem.

2.2.2.1 Classification of Metaheuristics Approach

There are a wide variety of metaheuristics and a number of properties along which to classify them [12][17][18][19].

Local search *vs*. Global search

One approach is to characterize the type of search strategy. One type of search strategy is an improvement on simple local search algorithms. A well known local search algorithm is the hill climbing method which is used to find local optimums. However, hill climbing does not guarantee finding global optimum solutions.

Many metaheuristic ideas were proposed to improve local search heuristic in order to find better solutions. Such metaheuristics include simulated annealing, tabu search, iterated local search, variable neighbourhood search, and GRASP. These metaheuristics can both be classified as local search-based or global search metaheuristics.

Single-solution *vs*. Population-based

Another classification dimension is single solution vs. population-based searches. Single solution approaches focus on modifying and improving a single candidate solution; single solution metaheuristics include simulated annealing, iterated local search, variable neighbourhood search, and guided local search. Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search; population based metaheuristics include evolutionary computation, genetic algorithms, and particle swarm optimization [22][24]. Another category of metaheuristics is Swarm intelligence which is a collective behaviour of decentralized, self-organized agents in a population or swarm. Ant colony optimization, particle swarm optimization, social cognitive optimization, and artificial bee colony algorithms are examples of this category. An example of memetic algorithm is the use of a local search algorithm instead of a basic mutation operator in evolutionary algorithms.

Hybridization and Memetic algorithms

A hybrid metaheuristic is one which combines a metaheuristic with other optimization approaches, such as algorithms from mathematical programming, constraint programming, and machine learning. Both components of a hybrid metaheuristic may run concurrently and exchange information to guide the search. On the hand, Memetic algorithms represent the synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search [26][28].

Nature-inspired metaheuristics

A very active area of research is the design of nature- inspired metaheuristics [21]. Many recent metaheuristics, especially evolutionary computation-based algorithms, are inspired by natural systems. Such metaheuristics include Ant colony optimization, particle swarm optimization, cuckoo search, and artificial bee colony to cite a few.

3. RECOMMENDATIONS FOR BETTER ESTIMATION

1. For accuracy any estimation technique should be applied through both the top down and bottom up approach.
2. Estimate the software size using a number of techniques, and then average these results to produce a combined estimate.
3. Estimation becomes more accurate as the development progresses.
4. Estimation should be based on several methods. If these do not return approximately the same result, then one has insufficient information available to make an estimate. Some action should be taken to find out more in order to make more accurate estimates.
5. Software projects require engineering, the process begins long before the product is designed – and it continues long afterward.
6. Meta -heuristics are general-purpose algorithms that can be applied to solve almost any optimization problem

4. CONCLUSION

Many estimation techniques, models, methodologies exist and are applicable in different nature of projects. None of the technique gives hundred percent accuracy but proper use and application of these techniques make estimation process smoother and easier. Active research in software estimation is always in process, especially making use of Artificial intelligence, regression and statistical models in software estimation. There has been a lot of research on the parameter evaluation done in COCOMO for software cost estimation and effort estimation number of optimisation techniques like ACO, Firefly, GA, PSO-FLANN have been used empirically to modify the conventional COCOMO model and its performance in terms of accuracy, prediction and error reduction is done and also analysis of optimization techniques like MOPSO, BAT, Firefly, Bee Colony Optimisation and Human Opinion Dynamics has been performed Organizations should tailor the existing methodologies to their requirements and customize these tools to bring the best out of them. The optimized models are assessed according to different evaluation criteria and compared with models optimized using other metaheuristic algorithms It would be important to work towards a hybrid approach that encompasses the best characteristics of different prediction schemes and work towards the high accuracy and significant error minimization [30][31][32][33].

5. PROPOSED WORK & FUTURE DIRECTION

Software estimation Techniques are critical, effective processes in software development and project management, many decisions stopped according to the results of the estimation, software estimation techniques needs extra efforts and cooperation from the academic researchers with a help from the industrial software development companies to achieve highly trusted models via exchanging expertise, models of development in addition to the software engineering best practices applied in the industrial software development company and the needed suitable data to formulate the metrics and cost models in software estimation process There is a need to improve the well known software estimation models as COCOMO, SLIM, PRICE-S models, most of these models require a historical data to forecast SE, COCOMO model for example was published by Barry W.Boehm's in 1981. Comparing organic and semi-detached COCOMO model models, it can be stated that use of the coefficients optimized by the GA in the organic mode produces better results in comparison with the results obtained using the current COCOMO model coefficients. After that A New Approach for Software Cost Estimation with Hybrid Genetic Algorithm and Ant Colony Optimization the proposed that the effective factors in estimation are using the

GA the test and using the ACO the training and better results are achieved in comparison to COCOMO model. But better models for estimation of the SCE of implementation and designing with more accurate estimation by combining other meta-heuristic algorithms.

6. REFERENCES

1. Forrest Shull, Carolyn Seaman, and Marvin Zelkowitz, "Victor R. Basili's Contributions to Software Quality", *IEEE Software Jnl* , pp. 16- 18, 2006.
2. Victor R. Basili, "Software Development: A Paradigm for the Future", *13th International conference on computer s/w and applications IEEE*, pp. 471-485, 1989.
3. ISO/IEC IS 15504 – 2: Software Engineering – Process Assesment – Part 2: Performing An Assesment, 2003.
4. ISO/IEC TR 9126-3: Software Engineering – Software Product Quality – Part 3: Internal Metrcis, 2000.
5. Roger S. Pressman, "Software Engineering: A Practitoner's Approach", 4 th Edition, McGraw Hill Inter. Editions, 1997.
6. Ed Yourdon, "Software Metrics", *Application Development Strategies*, Nov 1994.
7. Kemerer, C. F. "An Empirical Validation of Software Cost Estimation Models", *Comm. ACM 30*, pp. 416-429,1987.
8. Kemerer, C.F. and Porter, B.S. "Improving the Reliability of Function Point Measurement: An Empirical Study", *IEEE Transactions on Software Engineering*, Vol. SE-18, No. 11, pp. 1011-1024, 1992.
9. Mellis E. "Software Metrics. SEI Curriculum Module", *SEI- CM-12-1.1, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA*, 1998.
10. Magne Jorgensen Martin Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *Software Engineering, IEEE Transactions on 33*, vol 1 ,pp 33-53.,2007.
11. Sweta Kumari, and Shashank Pushkar, "Performance Analysis of The Software Cost Estimation Methods: A Review," *International J. Adv. Res.Comput. Sci. Software. Engg* ,pp.229-238,2013.
12. Sheta, Alaa F, "Estimation Of The COCOMO Model Parameters Using Genetic Algorithms For NASA Software Projects," *Journal of Computer Science* 2, vol.2 ,pp. 118- 123.2006.
13. Blum, C. and Roli, A. "Metaheuristics In Combinatorial Optimisation: Overview & Conceptural Comparision', *ACM Comput. Surv*., Vol. 35, pp. 268-308.2003.
14. Geem, Z. W., Kim J. K., and Loganathan, G. V.'A New Heuristic Optimisation: Harmony Search', Simulation, Vol. 76(2), pp. 60-68 2001.
15. Karaboga, D.'An Idea Based On Honey Bee Swarm For Numerical Optimisation', *Technical Report, Erciyes University* 2005.
16. Metaheuristic the wikipedia article from http://en.wikipedia.org /wiki/Metaheuristic 2010.
17. Moscato, P. " Evolution, Search, Optimisation, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". *Caltech Concurrent Computation Program report* 826 ,1989.
18. Talbi, E. G. Metahueristics: From Design to Implementation, Wiley 2009.
19. Kaswan, K.,Choudhary, S.,Sharma,K., "A New Classification And Applicability of Software Reliability Models" *Int. Jnl of Advance Research in Computer Sci. & Management Studies* Vol. 2, Issue 7, pp 101-104, 2014.
20. Galinina, A.,Burceva,O. ,Parshutin, S. "The optimization of cocomo model coefficients using genetic algorithms" *in Jnl of Info. Tech. & Management Sc*., pp. 46-51,2012.
21. Maleki I.,Ghaffari A.,Masdari, M. "A New Approach for Software Cost Estimation with Hybrid Genetic Algorithm and Ant Colony Optimization" *in Int. Jnl. of Innovation and Applied Studies*, vol 5 pp. 72-81 ,2014.
22. Nazeeh Ghatasheh1, Hossam Faris, Ibrahim Aljarah, Rizik M. H. Al-Sayyed "Optimizing Software Effort Estimation Models Using Firefly Algorithm" *Journal of Software Engineering and Applications* vol 8 pp. 133-142, 2015.
23. Jin-Cherng Lin, Han-Yuan Tzeng "Applying Particle Swarm Optimization To Estimate Software Effort By Multiple Factors Software Project Clustering" *IEEE Conference on Computer Symposium (ICS)*, pp. 1039-1042, 2010.
24. Xin-She Yang "A New Metaheuristic Bat-Inspired Algorithm" *NICSO 2010, Springer Berlin Heidelberg*, pp. 65-74, 2010.

25. Reddy. P"Software effort estimation using Particle Swarm Optimization with inertia weight"*International journal of software Engineering*, pp. 12-23, 2010.

26. Ruchi Puri, Iqbaldeep kaur, "Novel Meta-Heuristic Algorithmic Approach For Software Cost Estimation" *in Int.l Jnl. of Innovations in Engg. and Technology*, vol 5 pp.456-463, 2015.

27. Gupta, N., Sharma, K "Optimizing Intermediate COCOMO Model Using BAT Algorithm" *2nd International Conference on Computing for SustainableGlobal Development*, IEEE, pp. 1649 –1653,2015.

28. Yogesh Singh, Pradeep Kumar Bhatia, Omprakash Sangwan," ANN model for predicting software function point metric" ACM SIGSOFT Software Engineering Notes,vol.34 ,issue 1, pp 1-4,2009.

29. Y Singh, A Kaur, OP Sangwan," Neural model for software maintainability", Proceedings of International Conference on ICT in Education and Development, pp 1-11, 2004.

30. P. Kumar Singh, O.P Sangwan, A. Singh, A. Pratap," An Assessment of Software Testability using Fuzzy Logic Technique for Aspect-Oriented Software", International Journal of Information Technology and Computer Science, vol.7, issue 3, pp 18 ,2015.

31. P. Kumar Singh, O.P Sangwan, A.Kaur, Y. Singh," Application of neural networks in software engineering: A review", International Conference on Information Systems, Technology and Management,pp 128-137, 2009.

32. M.Mann, O.P Sangwan," Generating and prioritizing optimal paths using ant colony optimization", Computational Ecology and Software, Vol.5 issue 1, pp 1, 2015.

33. P. Kumar Singh, O.P Sangwan, A. Partap, A. Singh," Testability Assessment of Aspect Oriented Software Using Multicriteria Decision Making Approaches", World Applied Sciences Journal, Vol 32, Issue 4, pp 718-730, 2014.