DOI: 10.21917/ijms.2015.0027

AN ENHANCED CLIENT CENTRIC SOFTWARE DEVELOPMENT LIFE CYCLE MODEL WITH COST AND EFFORT ESTIMATION

Vanshika Rastogi [1], G. Swetha[2], E. Anantha Lakshmi[3] and M. Pauline[4]
[1,2,3]*Department of Information Science and Engineering, MVJ College of Engineering, India*
E-mail: [1]rastogi.vanshika21@gmail.com, [2]swetha.lahari@gmail.com, [3]ananthaeli@gmail.com
[4]*Department of Computer Science and Engineering, MVJ College of Engineering, India*
E-mail: pmariasundaram@yahoo.com

Abstract

In today's world of software development, there exist many Software Development Life Cycle (SDLC) Models. SDLC model gives a theoretical guideline for the development of the software. Each model has its own characteristics, limitations and working environment. With rapid advances in communication and information technology, organizations have to deliver high-quality software, solutions at low Cost and satisfy the client needs. Employing proper SDLC model allows the project managers to regulate entire development strategy of the software. In the paper, author proposes a model for software development that incorporates prototype and spiral model, defining an output with each stage. The proposed model fulfills the client's requirement, providing high quality product with less Effort and Cost. The proposed model allows client and developer to interact with each other in order to understand and implement requirements in an organized way. The proposed model considers one of the real time applications (HR) as a case study, and evaluates the Effort and Cost in terms of enhanced function point and lines of code. A Comparison of Effort estimation and Cost estimation of various existing models and the proposed model is done. The proposed model is Effort and Cost effective. The proposed model is more towards client centric SDLC model.

Keywords:

Software Development Life Cycle (SDLC), Client Satisfaction, Effort, Function Point (FP), Kilo Lines of Code (KLOC,) Cost Estimation

1. INTRODUCTION

The authors propose Software Development Life Cycle Model which aims at client satisfaction with an efficient Effort and Cost estimation. The intent of a SDLC process is to help produce a high quality product with less Effort and that is Cost-effective. To develop a software product, industries deploy SDLC model that suits their needs. The SDLC methodology consists of the following stages: They are Analysis phase (requirements and design), construction phase, testing phase, release phase and maintenance (response) phase.

LOC is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code. SLOC is used to predict the amount of Effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced. Function points [FP] are the unit of measure to express the amount of business functionality provided to the user by the software. FP is an indirect measure of software size based on external and internal application characteristics as well as application performance. FP is used for estimation during the early stages of the project and later to measure the actual size of the application delivered. FP can be used to Estimate the Effort or Cost required to design, code and test the software. FP is used to predict the errors that will be encountered during testing and forecast the number of components and/or the number of projected source lines in the implemented system. In the first phase; a comparative study and analysis is made on various existing SDLC models. In second phase, proposed SDLC model is discussed with its working principle. The third phase presents the Effort and Cost estimation of the proposed work. A comparison of the Effort and Cost for various existing models and the proposed the model is done.

2. RELATED WORK

Software engineering is the study and application of engineering to the design, development and maintenance of software [1]. The discipline of software engineering was created to address poor quality of software, get projects exceeding time and budget under control and ensure that software is built systematically, rigorously, measurably, on time, on budget and within specification. This engineering discipline describes how software should be developed [2]. In regard to the development of software, there exists many models, that work into distinct phases (or stages) containing activities with the intent of better planning and management in order to develop the software [3]. Common methodologies include waterfall, prototyping, iterative and incremental development, spiral development, rapid application development, extreme programming and agile methodology [4]. Different SDLC models and their pros and cons are presented in [5]. In paper [6], authors have analyzed and compared SDLC models. Authors have proposed, comparing models mathematically would be feasible [7]. A model that merges basic phases of SDLC and release management is proposed thus increasing the effectiveness of the SDLC model, establishing which role has to do what and when in various stages[8]. Client's benefits in terms of Cost as well as satisfaction also need to be considered. Large numbers of software projects do not meet the client expectations in terms of Functionality, Cost or delivery schedule. In [9] author proposes a model, SDLC 2013, which works towards client satisfaction process. The work in this paper proposes SDLC model 2015 to overcome the short-comings; above all, it ensures that the client is satisfied. The authors propose a SDLC model to develop a product or software. The model starts with clients requirement developed using prototype model, followed by the spiral model development [10]. The combinations of the two different SDLC models help to develop a software or product with clients' involvement throughout the development, ensuring to satisfy the client and also benefiting the organization. The paper evaluates

the models efficiency using basic software metrics [11]. The proposed models Effort and Cost are estimated using enhanced function point and Cost estimation proposed in [12]. Function point (FP) counting passes through an adjustment phase [14, 15]. This phase consists of scoring a group of general systems characteristics (GSC) that rate the general functionality of the application being counted, from the GSC, the value adjustment factor VAF determined[16,17]. Enhanced function point can then be calculated [18]. Analysis of Empirical Software Effort Estimation Models is presented in [19]. Estimation using expert judgment is better than models [20]. Existence of a consistently applied process is important and a prerequisite for a successful measurement program in case of different environments [21]. The calculated FP is then used to estimate the Cost [22, 23]. Intermediate COCOMO is used for Cost calculation. Cost factors are chosen based on individual development environment, which is crucial for the accuracy of Cost estimation [18].

3. SYSTEM OVERVIEW

The proposed SDLC-2015 model is designed in a way that allows the client to communicate with the developer to get satisfied with the implementation of the requirements. The proposed model aims at developing an efficient software product that satisfies the client, with less Effort and Cost effective. Requirement analysis phase is very crucial and important in any software development life cycle. Improper analysis leads to client dissatisfaction that may have effect on time, Effort and Cost of the software.

The proposed model starts with a Client interacting with Team A: Requirement Analysis/gathering team, then with Team B: Technical Team. Client has to be involved in the software development process to order to achieve client satisfaction. But this should not affect the Schedule, Effort and Cost of the organization as well as hinder the performance of the product. In order to achieve this, our model helps client involvement in a well-planned - ordered manner throughout the product development. Initially, Client and team A will work together understanding the requirements of the client. This is very important and crucial in software product development. Here the team analyses the requirement, with the help of developer, tester as well as management member, they conclude if the product can be implemented or not. During this phase, the Effort and Cost will also be calculated, that helps client to be aware of the Cost and schedule of the final product delivery.

Simultaneously, requirement team passes the requirement to design team, who will search if any existing similar product exists, to show it as a prototype with fewer or no modifications made or if no similar software exists, Team A builds a new prototype to match the client requirement and gives it to client for evaluation. This makes the client to analyze how the product might be and also to better understand his requirements. Once the client agrees on the prototype, the requirements implemented in the prototype will be freezed so as to reuse prototype. Now the client is aware of the kind of product he will be delivered, the Cost and the time it takes to deliver the product. Management can also estimate the Effort and Cost required to develop the product.

The product now goes to next phase, with Team B, where client along with requirement team member, design team, developers, testing team will be involved. The development here follows spiral model. Requirement team chooses the requirement to be implemented based on clients request, identifies, analyses the risk and passes on this to design team. This team designs the module/unit in compliance with existing prototype, pass this to development team. Development team codes and passes for further testing. On successful testing, this is then integrated with prototype tested and handed to client for check. This process repeats until the product is developed. Hence, the involvement of client in each phase guarantees client satisfaction as well as avoid the many changes from the client that might affect product Effort and Cost by freezing the prototype. Thus our proposed model SDLC-2015, calculates Effort and Cost, also a comparative study of the Cost and Effort of various existing models and the proposed models is done.

4. PHASES OF SDLC MODELS

Software Development Life Cycle (SDLC) is a descriptive and diagrammatic representation of the software life cycle. It represents all the activities required to make a software product. It consists of Requirement Analysis phase, Design phase, Coding phase, Testing phase and Maintenance phase.

4.1 REQUIREMENT ANALYSIS PHASE

Requirements analysis is critical to the success of a systems or software project. The first step is to perform a thorough analysis of the client's current situation, careful to define the situation as precisely as possible and obtain a clear understanding of what the software product must do. During the requirements phase the team must:

- Attempt to determine the real needs of the client
- Avoid blindly taking statements about the client's wants e.g. a wish list
- Recognize the client is not always conscious of all the needs
- Overcome any lack of computer-literacy on the part of the client i.e. bridge the technical divide between developer and client
- Correctly interpret client's requests even if not stated in the best way possible

4.2 FEASIBILITY STUDY PHASE

This is the next phase of the development life cycle. It Includes analysis of project requirements in terms of input data and desired output, processing required to transform input into output, cost-benefit analysis and schedule of the project. The feasibility analysis also includes the technical feasibility of a project in terms of available software tools, hardware and skilled software professionals. At the end of this phase, a feasibility report for the entire project is created that details the risks, resources required, Estimate Effort, Cost and time required, legal requirements if any etc. At the end, an updated Software Requirement Specification [SRS] document is presented.

4.3 DESIGN PHASE

Design includes translation of the requirements specified in the SRS into a logical structure that can be implemented in a programming language. There are two design approaches the traditional design approach and the object oriented design approach.

The traditional design approach consists of two different activities. First, a structured analysis of the requirement specification is done. It involves preparing a detailed analysis of the different functions to be supported by the system and the identification of the data flow among the different functions. This is followed by a structured design activity. The output of the design phase is a design document that acts as an input for all the subsequent SDLC phases.

Object oriented design is a technique in which various objects that occur in the problem domain and the solution domain are first identified and the different relationships that exist among these objects are also identified.

4.4 CODING PHASE

Coding is the actual implementation of the design specified in the design document into executable programming language code. The output of the coding phase is the source code for the software that acts as input to the testing and maintenance phase.

4.5 TESTING PHASE

Testing includes detection of errors in the software. The testing process starts with a test plan that recognizes test-related activities such as test case generation, testing criteria and resource allocation for testing. The code is tested and mapped against the design document created in the design phase. The output of the testing phase is a test report containing errors that occurred while testing the application or any deviations with respect to the requirement document.

4.6 MAINTENANCE PHASE

Maintenance of the development product includes implementation of changes that software might undergo over a period of time or implementation of new requirements after the software is deployed at the customer location. The maintenance phase also includes handling the residual errors that may exist in the software even after the testing phase.

5. SDLC MODELS

There exist many SDLC models, which are followed to develop a software product. One software development methodology framework is not necessarily suitable for use by all projects. Each of the available methodology frameworks are best suited to specific kinds of projects, based on various technical, organizational, project and team considerations. Typically an approach or a combination of approaches is chosen by management that benefits the organization or chosen by a development team, based on the available recourses. Also, organization chooses to use a particular model, based on various client requirements, man power, Effort required, time and Cost.

5.1 WATERFALL MODEL

Water fall model is the most well know software development life cycle model. The waterfall model is a sequential design process. This model specifies what the system is supposed to do (i.e. define the requirements) before building the system (i.e. designing) and plans how components are going to interact (i.e. designing) before building the component (i.e. coding). In this model, one should move from one phase to next phase only when its preceding phase is reviewed and verified.

Features:

- Simple and easy to manage.
- Sets requirements stability.
- Each phase has specific deliverables and a review process
- Phases are processed and completed one at a time, no overlapping of phases.
- Used for smaller projects
- Works well when quality is more important than Cost or schedule.

Drawbacks:

- Once the product enters to development stage, product is developed based on the requirements recorded in the first phase. Thus there is no formal way to make changes to the project in the later stages, if there is any change in the requirements or more information becomes available to the project team.
- Risk is high and therefore high uncertainty.
- Delays in discovery of serious errors.
- Not a good model for complex projects.

5.2 PROTOTYPE MODEL

Prototype model can be used in the projects where the requirements are unstable or have to be clarified. Useful to provide short-lived demonstrations for the client to understand requirements better. Also used for new product development, whose requirements are new.

Features:

- It provides a better system to users, as the prototype, gives a better understanding of the model, user is in need.
- Interaction with the prototype stimulates awareness of additional needed functionality or any changes if needed.
- Errors can be detected much earlier
- It saves the time and Cost.

Drawbacks:

- Generally, prototypes are throw-away.
- Many changes in the prototype may hinder the functionality, increase complexity, Cost, time and Effort.
- Tendency to abandon structured program development for “code-and-fix” development
- Not suitable for large applications.

5.3 SPIRAL MODEL

The spiral model is an evolutionary software process model which is a combination of an iterative nature of prototyping and controlled and systematic aspects of traditional waterfall model. It is suitable for development of technically challenging software products that are prone to several kinds of risks. It allows for incremental releases/refinement of the product. The spiral model is also called as "meta-model", since it encompasses all other lifecycle models.

Each phase in this model is split into four sectors. The first quadrant identifies the objective of the phase and the alternative solutions possible for the phase under consideration. During the second quadrant, the alternative solutions are evaluated to select the best solution possible. For the chosen solution, the potential risks are identified and dealt with by developing an appropriate prototype. Activities during the third quadrant consist of developing and verifying the next level of the product. The fourth quadrant activities concern reviewing results of the stages traversed so far, with the customer and planning the next iteration around the spiral.

Features:

- Good for large and mission critical projects
- High amount of risk analysis
- Software is produced early in the software life cycle.
- Uses rapid prototyping tools and produces prototypes making it easy for client to understand product better.
- Critical and high-risk functions are given importance and developed first
- Proper control over Cost, time if followed efficiently

Drawbacks:

- Costly and Cost and time estimation will be difficult.
- Model completely works on risk identification, its projection, assessment and management and hence risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects

5.4 SDLC 2013

SDLC 2013 is a model, developed to provide client satisfaction. The client is involved throughout the development process. The coordinator acts as mediator between the technical team and the client. This model develops or produces the existing prototype that matches client specification-known as matching software, for understanding the requirements efficiently from the client in order to estimate Cost, schedule and Effort.

Features:

- Client gets better understanding about his requirements and product going to be developed.
- Client gets the look and feel of the application.
- Matchmaker team reduces the work of re-building the prototype.
- Flexibility for any change in the software to meet the client requirement is made easy.

Drawbacks:

- Involvement of Client through-out the process, without freezing at-least the minimum requirement, leads to often changes.
- Often changes in the software might also disturb the functionality of the prototype agreed by the client.
- Due to this, more Effort, re-work, time as well as Cost might increase.

6. PROPOSED MODEL

In the proposed model, client interacts with Requirement gathering/analysis team. This team comprises of members from design team - a Lead Architect, Technical Lead from development, Test Lead from testing also headed by a Team Manager. The client interacts with Requirement Analysis/gathering Team and discusses his/her requirements. This team analyses the requirement, finds if any software with similar requirements exists and gives it to the client as prototype for the client to understand and get the insight of his/her product would be. This also helps client to understand the requirements and future needs. If there does not exist any matching software, design team generates a design prototype, for the basic functionality and a prototype is built for the client. Requirement analysis and gathering team also does the Effort, Cost and schedule estimation and planning. Upon client's approval, the product is carried for further development by technical team.

Technical Team consists of members from developing team, software design members, say Architects, Testers as well as the requirement analyzers. Technical team follows spiral model for the development of the client's product. This team takes up one requirement at a time, analyses risk associated, prioritizes the risk, looks for any alternatives if needed and then does a design for the requirement which is later followed by developers to build the product. Once the module is developed, it is handed to testers, to map it with the client's requirement and tested for various inputs. This is then integrated with the prototype and given to the client for further check. Care has to be taken so that the initial prototype does not change, as well as the client's further requirement changes also will not change the agreed prototype.

6.1 REQUIREMENT GATHERING/ANALYSIS TEAM

This team comprises of people who have knowledge of each and every phase of software development life cycle. The team consists of people with knowledge of operating systems, applications to be used, members from design, testing and management teams. This team gathers the requirement from the client does the feasibility study and accesses risk. After finalizing the requirements, it is further passed for a quick design. The design team first goes through the requirements and then search for any existing software that matches with the client's requirement. If something similar already exists, that will be given to the client for his evaluation through the requirement analysis/gathering team. If there is no matching

product available, then a new prototype is built and given to client for evaluation.

The prototype is always given along with a time slot for the client to evaluate. If the client wants to have certain changes in the requirements, they will be taken by the team and again the prototype will be designed. Once the client is satisfied with the prototype, then the agreed/implemented requirements will be freezed i.e. no further changes can be made in the requirements.

6.2 TECHNICAL TEAM

This is a team of people who are fully expertise in the various fields of software development. It consists of people from requirement team, design, development and testing. It is this team who is actually going to satisfy the client. Initially this team of design will have a detailed discussion with the requirement analysis team before starting up their work.

They first understand the requirements, access the risks, analyze and then design the requirement, in accordance with existing prototype, pass on to development team. Development team builds the small units/modules and passes for testing for further check. This module is then given to client for a check. Upon agreeing, it is integrated with existing prototype. This process is carried in a spiral manner. In each requirement development, client along with technical members are involved.

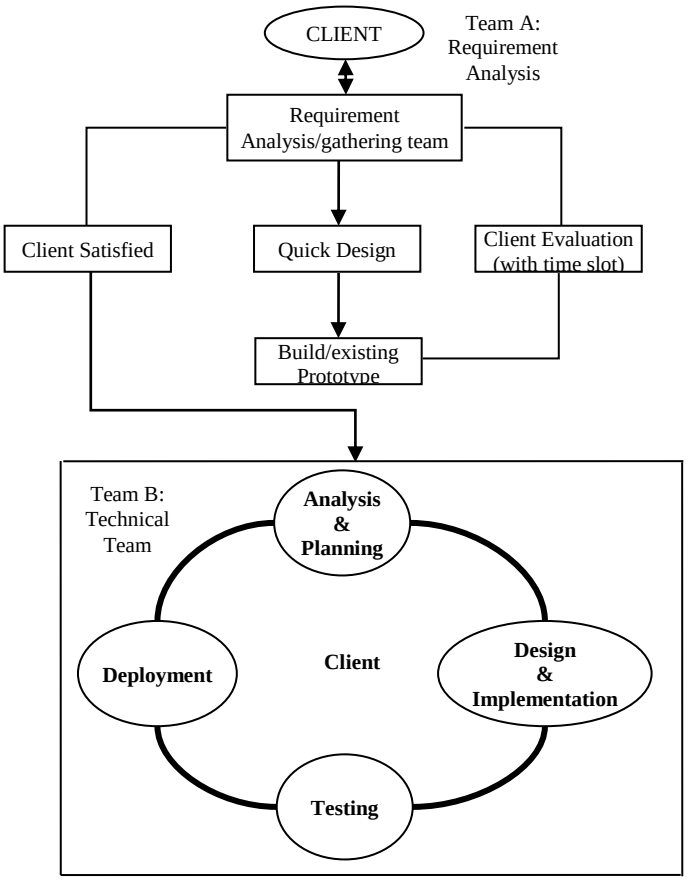


Fig.1. Proposed SDLC 2015

7. EFFORT AND COST ESTIMATION

The Effort and Cost estimation of our proposed model is evaluated. The proposed model considers the fuzzification of the function points to calculate Effort. Fuzzification considers all the six key quality attributes, they are Functionality, Reliability, Usability, Efficiency, Maintainability and Portability and are grouped into four fuzzy sets to estimate quality Effort and Cost. Cost and Effort estimation is required to design code and test the software. It is also used to predict the number of errors, forecast the number of lines of code to be projected, number of personnel required to implement the system.

7.1 LINES OF CODE

LOC is presented as a measurement technique for quantifying the size of a software product. LOC is more of a measurement technique than a counting technique. There are many ways of obtaining the LOC of a program without actually counting program lines of code. The steps for calculating Lines of codes are:

- Each Statement (executable or declarative) is counted as one line.
- Comments are excluded from the count.
- For languages that use delimiters each delimiter corresponds to one statement.

7.2 FUNCTION POINT

The function point metric (FP) proposed by Albrecht can be used effectively to estimate the Cost or Effort required to design, code and test the software, predict the number of errors that will be encountered during testing and forecast the number of components and/or the number of projected source lines in the implemented system.

The basic steps to calculate Function Point metric:

- Count total is calculated using information domain and weighing factors.
- The Value added factor is based on the responses to the 14 characteristics, each involving a scale from 0 to5.
- Function point is the product of Count Total and the Value added factor.

Thus Function points (*FP*) provide a measure of the functionality of a software product and can be calculated through the equation:

$$FP = \text{count-total} \times [0.65 + 0.01 \times \sum Fi]$$

where, count-total is the total of weighted input/output characteristics and *Fi* is the summation of fourteen ranked factors. Below are the proposed model's factor values for the HR application:

System Complexity

Data Communication	3
Distributed Data Processing	2
Performance	3
Heavily used configuration	2

I/O Complexity

Transaction rate	2
On-line data entry	3
End User Efficiency	4
On-line update	3

Application Complexity

Complex Processing	2
Reusability	3
Installation Ease	2
Operational Ease	2
Multiple sites	3
Facilitate Change	4

Quality Complexity

Quality of requirements (for our model)	0.5

Estimated FP = Count Total × [0.65 + 0.01 × ∑ (Fi)]

FP Estimated for Existing (HR)

= 88 × [0.65 + 0.01 * 38] = 90.64 FP

FP for the proposed model =

88 × [0.65 + 0.01*8.67] = 64.83 FP

7.3 COST ESTIMATION USING INTERMEDIATE COCOMO FOR HR APPLICATION

COCOMO (COnstructive Cost MOdel) is empirical Cost estimation model that is self-sufficient in providing a somewhat a clear picture in mathematical terms, regarding the software being developed. The Intermediate COCOMO Equation is given by

$$E = a \text{ KLOC} \wedge b * EAF$$

where, a and b are the domain constants of the intermediate COCOMO Model, these formula link the size of the system, domain constants and Effort multipliers [EM] to find the Effort to develop a software system.

KSLOC = FP * Multiplication Language Factor

KSLOC (Using Albrecht method) = 90.64 * 29

= 2628.56/1000 = 2.6 KSLOC

KSLOC (For the proposed Model) = 64.83 * 29

= 1880.07/1000 = 1.8 KSLOC

Effort = a * KSLOC ^ b * EAF.

As the HR application is a semidetached project, we consider a = 3.0 and b = 1.12.

By selecting minimal ratings for product and computer attributes and maximum ratings for Personal and Project attributes. Effort multiplier is 0.072.

Effort = 3.0 * 1.8 ^ 1.12 * 0.072 = 0.41 PM

8. EXPERIMENTAL SETUP AND RESULTS

In order to estimate the Effort and Cost of our proposed work, authors have considered data for HR application.

Table.1. comparison of FP and KSLOC for various models

	Waterfall	**Prototype**	**Spiral Model**	**SDLC 2013**	**Proposed model**
FP	89.76	88.88	86.24	73.04	64.83
KS LOC	2.60	2.57	2.50	2.118	1.8
Effort (Person month)	0.63	0.62	0.60	0.49	0.41
Time (month)	2.12	2.11	2.09	1.94	1.82
People (count)	0.29	0.29	0.28	0.25	0.22

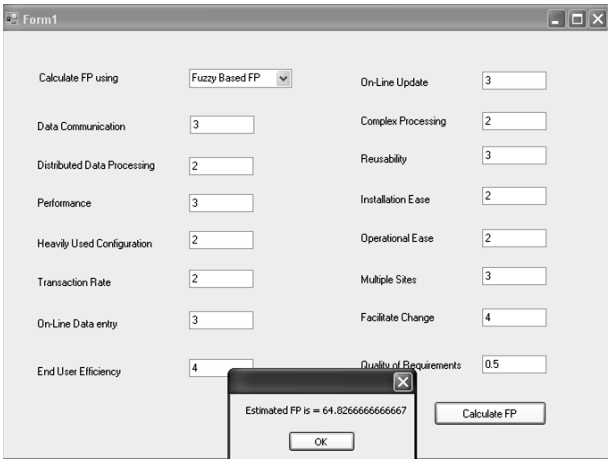
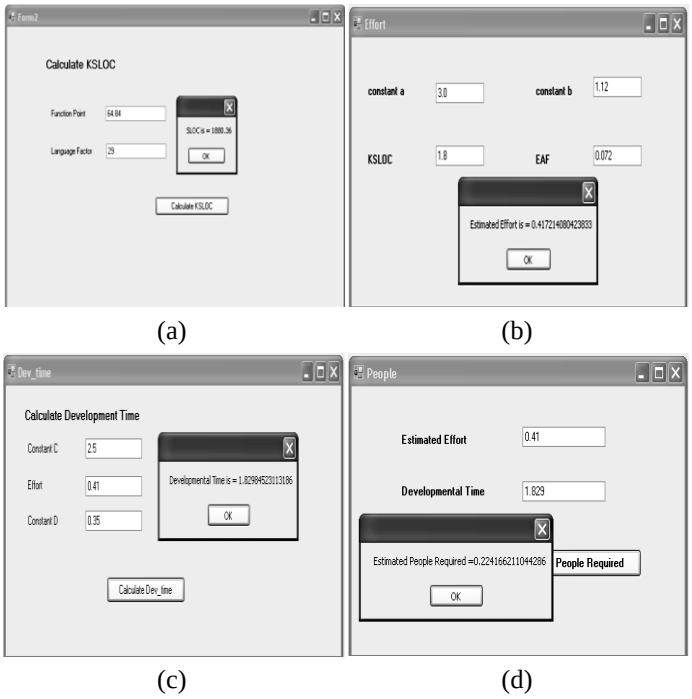


Fig.2. Calculation of FP using Fuzzification of FP

An application is written in C#.NET in order to evaluate the proposed model. Function point is calculated using Albrecht's method for waterfall model, Prototype model, Spiral model, SDLC 2013 and fuzzification of input values for the proposed model. A comparative analysis is produced in the Table.1 above.

The Fig.2 shows the simulation results of the HR application in order to estimate FP. The Fig.3(a) shows KSLOC estimate, Fig.3(b) calculates the Effort estimate, Fig.3(c) shows Developmental time calculation, Fig.3(d) estimates the people required. The experimental setup calculates function point and KSLOC using Albrecht's method and fuzzification method for various SDLC models. The results show that the proposed model estimates Cost and Effort better (with An Enhanced Model to Estimate Effort, Performance and Cost of the Software Projects [16]) thus providing better Effort and Cost model than others.



(a) (b)

(c) (d)

Fig.3. (a) Calculation of KSLOC, (b) Calculation of Effort, (c) Calculation of Developmental Time, (d) People Required

9. CONCLUSION AND FUTURE WORK

The paper can be summarized as the creation of a new SDLC model - SDLC-2015 which is capable of developing the software with lesser Effort, within time and budget.

The proposed work considers three primitive primary software engineering metrics, i.e., LOC, FP, Developmental time and PM to evaluate the models Effort and Cost. This model uses fuzzy logic based approach, to estimate the Effort in the software product development. Calculated FP from our work is used as input to COCOMO Intermediate for the Cost estimation in terms of KSLOC. This is very important for estimating the Effort required, Cost, software performance, duration and schedule of the project.

A comparative study for Effort and Cost estimation of the existing model and our proposed model is done considering a software project i.e., HR application as an example. Thus our work proposes and analyses the Effort and Cost of various models with respect to proposed model - SDLC 2015. In future the work can be extended to estimate Cost using enhanced methods. Performance of the proposed model can be evaluated in future.

REFERENCES

[1] Phillip A. Laplante, "*What Every Engineer Should Know about Software Engineering*", Taylor & Francis Group, 2007.

[2] "Selecting a development approach", Office of Information Services, pp. 1-10, 2008. Available at https://www.cms.gov/research-statistics-data-and-systems/cms-information-technology/xlc/downloads/selectingdevelopmentapproach.pdf

[3] Irwin Brown, "*Proceedings of the International Conference on Information Management and Evaluation*", University of Cape Town, South Africa: Academic Publishers, 2010.

[4] Roger S. Pressman, "*Software Engineering, a Practioners Approach*", 6th Edition, McGraw Hill, 2007.

[5] Vanshika Rastogi, "Software Development Life Cycle Models Comparison, Consequences", *International Journal of Computer Science and Information Technology*, Vol. 6, No. 1, pp. 168-172, 2015.

[6] Seema and Sona Malhotra, "Analysis and Tabular Comparison of Popular SDLC Models", *International Journal of Advances in Computing and Information Technology*, Vol. 1, No. 3, pp. 277-286, 2012.

[7] Vishwas Massey and K.J. Satao, "Comparing Various SDLC models and the New Proposed Model on the basis of Available Methodology", *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 2, No. 4, pp. 170-177, 2012.

[8] Vishwas Massey and K.J. Satao, "Evolving a New SDLC model incorporated with Release Management", *International Journal of Engineering and Advanced Technology*, Vol. 1, No. 4, pp. 25-31, 2012.

[9] Naresh Kumar, A.S. Zadgaonkar and Abhinav Shukla, "Evolving a New Software Development Life Cycle Model SDLC-2013 with client satisfaction", *International Journal of Soft Computing and Engineering*, Vol. 3, No. 1, pp. 216-221, 2013.

[10] Barry W. Boehm, "A Spiral Model of Software Development and Enhancement", Vol. 21, No. 5, pp. 61-72, 1988.

[11] D.S. Kushwaha and A.K. Misra, "Cognitive Software Development Process and Associated Metrics - A Framework", *Proceedings of 5th International Conference on Cognitive Informatics*, pp. 255-260, 2006.

[12] M. Pauline, P. Aruna and B. Shadaksharappa, "An Enhanced Model to Estimate Effort, Performance and Cost of the Software Projects", *ICTACT Journal of Soft Computing*, Vol. 3, No. 3, pp. 524-533, 2013.

[13] M.J. Basavaraj and K.C. Shet, "Software Estimation using Function Point Analysis Difficulties and Research Challenges", *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*, pp. 111-116, 2007.

[14] D. Longstreet, "*Function Points Step by Step*", Longstreet Consulting Inc., 1999.

[15] C.J. Lokan, "An Empirical Analysis of Function Point Adjustment Factors", *Information and Software Technology*, Vol. 42, No. 9, pp. 649-659, 2000.

[16] Amr Kamel, Galal H. Galal-Edeen and Hanan Moussa, "Lessons Learned from Building an Effort Estimation Model for Software Projects", *International Journal of Software Engineering*, Vol. 3, No. 2, pp. 71-86, 2010.

[17] Valerie Marthaler, "*Function Point Counting Practices Manual Release 4.1.1*", International Function Point User's Group, 2000. Available at http://perun.pmf.uns.ac.rs/old/repository/research/se/functionpoints.pdf

[18] M. Pauline, P. Aruna and B. Shadaksharappa, "Fuzzy-Based Approach Using Enhanced Function Point to Evaluate the Performance of Software Project", *The IUP Journal of Computer Sciences*, Vol. 6, No. 2, pp. 20-33, 2012.

[19] Saleem Basha and P. Dhavachelvan, "Analysis of Empirical Software Effort Estimation Models", *International Journal of Computer Science and Information Security*, Vol. 7, No. 3, pp. 68-77, 2010.

[20] Magne Jorgensen, "Practical Guidelines for Expert-Judgment-Based Software Effort Estimation", *IEEE Software*, Vol. 22, No. 3, pp. 57-63, 2005.

[21] Frank Niessink and Hans van Vliet, "Two Case Studies in Measuring Software Maintenance Effort", *Proceedings of International Conference on Software Maintenance*, pp.76-85, 1998.

[22] Samuel Lee, Lance Titchkosky and Seth Bowen, "*Software Cost Estimation*", Department of Computer Science, University of Calgary, Available at http://www.computing.dcu.ie/~renaat/ca421/report.html.

[23] M. Pauline, P. Aruna and B. Shadaksharappa, "A Cost Model for Estimation of the Software Developed", *Proceedings of International Conference on Communication, Computation, Control and Nanotechnology*, pp. 762-764, 2010.