Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Comparing Various SDLC Models And The New Proposed Model On The Basis Of Available Methodology

Vishwas Massey
M.Tech (SE) Scholar, Rungta College of Engg. & Tech.,
Bhilai, Chhattisgarh-490024,India
vishwasmassey@yahoo.com

Prof. K.J.Satao
Professor & HOD, Rungta College of Engg. & Tech.
Bhilai, Chhattisgarh -490024,India

Abstract— There are various SDLC models widely accepted and employed for developing software. SDLC models give a theoretical guide line regarding development of the software. Employing proper SDLC allows the managers to regulate whole development strategy of the software. Each SDLC has its advantages and disadvantages making it suitable for use under specific condition and constraints for specified type of software only. We need to understand which SDLC would generate most successful result when employed for software development. For this we need some method to compare SDLC models. Various methods have been suggested which allows comparing SDLC models. Comparing SLDC models is a complex task as there is no mathematical theorem or physical device available. The essence of this paper is to analyse some methodologies that could result in successful comparison of the SDLC models. For this we have studied various available tools, techniques and methodologies and have tried to extract most simple, easy and highly understandable method for comparing SDLC models.

Keywords— CoCoMo-81, COSYSMO, Putnam Model, Release management, SDLC models.

I. Introduction

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow for developing software. Understanding the basic concepts of software development methodologies is necessary to enable evaluation of best software development life cycle (SDLC) methodology. All software projects go through the phases of requirements gathering, business analysis, system design, implementation, and quality assurance testing[1]. Employing any SDLC model is often a matter of personal choice entirely dependent on the developer. Each SDLC has its strengths and weaknesses, and each SDLC may provide better functionalities in one situation than in another. SDLC models vary greatly in their scope, end-user involvement, risk assessment, and quality control. Then there arises the question which model will provide what functionalities and under what expectations. One life cycle model theoretically may suite particular conditions and at the same time other model may also looks fitting into the requirements but one should consider trade-off while deciding which model to choose[2].

II. COMMONLY USED MODELS

The discovery, design, development and delivery of information systems are often linked together in a process labelled the Systems Development Life Cycle (SDLC). There are different models which follow these basic steps like *Waterfall, Prototyping , Spiral , Iterative and incremental development, Agile development, Rapid ,* Rapid prototyping, Evolutionary Model. In this paper we are describing only three of them which are mostly used.

A. *Waterfall Model*

Waterfall model was proposed by Royce in 1970 which is a linear sequential software development life cycle (SDLC) model. The various phases followed are **r**equirements analysis, design, coding, testing and implementation in such a manner that the phase once over is not repeated again and the development does not move to next phase until and unless the previous phase is completely completed**.** The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance[3].

B. *Incremental Model*

Incremental model is the advancement of the waterfall model. The phases of waterfall model are employed in such a manner that the result of any of the increment is used back as the input for the next increment. Thus with each increment there are some client's feedback that is used in getting the next incremental product. Thus with each ongoing increment the functionality of the core product gets enhanced. The incremental Model is an evolution of the waterfall model, where the waterfall model is incrementally applied. The series of releases is referred to as “increments”, with each increment providing more functionality to the customers. After the first increment, a core product is delivered, which can already be used by the customer. Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly. This process continues, with increments being delivered until the complete product is delivered [4].

C. *Spiral Model*

The spiral model[2,5,6] was defined by Barry Boehm in his 1988 article A Spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration matters. As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with the client (who may be internal) reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project. The different phases of spiral model are widely used for industrial real word complex software. This is the most advantageous and practical model of all the SDLC models. The whole model is an iterative spiral steps that is continuously repeated over and over time to generate the actual software components with each spiral. Thus help in reducing the complexity of the software being developed. The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spirals, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral. Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions

III. NEW PROPOSED SDLC MODEL

The new proposed model is developed by incorporating Release Management within the scope of the SDLC basic phases like analysis, design, coding, testing and maintenance.

Release Management is the concept which is quite new in the field of Software Engineering. The concept of release management derives itself from the core concept of project management employed in Software Engineering. Software how-so-ever efficient and effective cannot be considered commercially successful until and unless the software remains in the market for sufficiently long duration, in order to recover the cost that incurred during development and deployment of the software.

The **release management[7]** process is a relatively new but rapidly growing discipline within software engineering of managing software releases. As software systems, software development processes, and resources become more distributed, they invariably become more specialized and complex. Furthermore, software products (especially web applications) are typically in an ongoing cycle of development, testing, and release. Add to this an evolution and growing complexity of the platforms on which these systems run, and it becomes clear there are a lot of moving pieces that must fit together seamlessly to guarantee the success and long-term value of a product or project.

The need therefore exists for dedicated resources to oversee the integration and flow of development, testing, deployment, and support of these systems. Although project managers have done this in the past, they generally are more concerned with high-level, "grand design" aspects of a project or application, and so often do not have time to oversee some of the more technical or day-to-day aspects. Release managers (aka "RMs") address this need. They must have a general knowledge of every aspect of the software development process, various applicable operating systems and software application or platforms, as well as various business functions and perspectives.

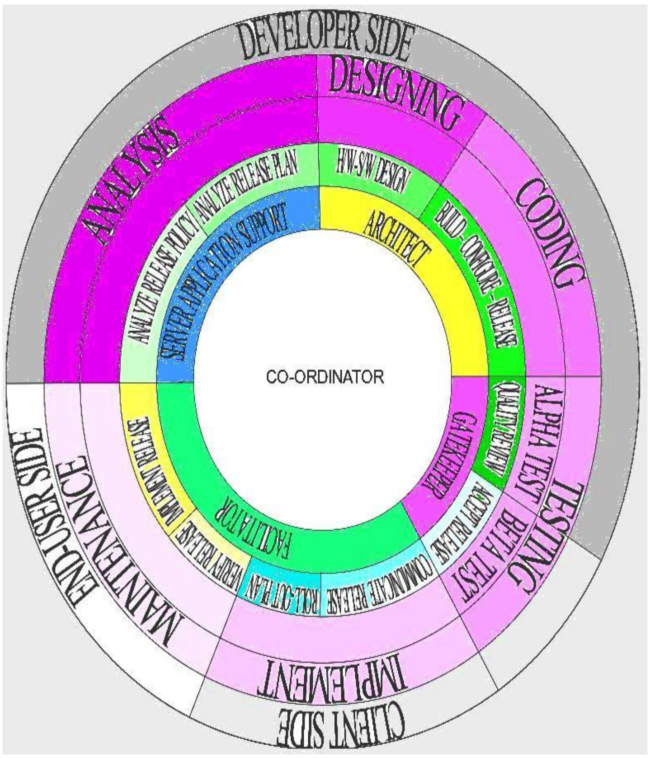

Figure. New praposed model – SDLC ‘VISHWAS’

A. *Model Architecture*

The architecture of the model is based on the following components

· Release Manager as centralised core functional controllers for SDLC phases

· Developer side consisting of all the people responsible for technical aspects
· Client side consists of interested organisation involved in purchase of software
· End user side is the actual users of the systems.

1. Release Manager

Release managers[8,9] (aka "RMs") address the need for dedicated resources to oversee the integration and flow of development, testing, deployment, and support of these systems. Although project managers have done this in the past, they generally are more concerned with high-level, "grand design" aspects of a project or application, and so often do not have time to oversee some of the more technical or day-to-day aspects. RMs must have a general knowledge of every aspect of the software development process, various applicable operating systems and software application or platforms, as well as various business functions and perspectives. According to the various functions to be performed, the release manager can be classified as:

i. **Coordinator**: The heart of the model is the “Coordinator” which coordinates the functions of different release managers. He coordinates all the phases of the software development process. He is responsible to coordinate disparate source trees, projects, teams and components. Coordinator makes communication between the developers and client via other release managers.

ii. **Server-Application-Support Engineer:** Release manager responsible for analyzing the troubleshoots problems with an application not typically at a code level but more at initial analysis level. He is even responsible for analyzing release policy and release plan.

iii. **Architect**: Release manager responsible to identify, create and/or implement processes or products to efficiently manage the release of code. An architect is a person trained in the planning, design and oversight of the construction of any tangible and intangible products. Architect channels communication between designers and coders to enable them for understanding the requirements of the client.

iv. **Gate-Keeper**: Release manager engaged in collecting the test results from the testers and communicating it back to coders and designers via coordinator. He ensures that the tests conducted are in accordance with the client requirements and is satisfactory successful.

v. **Facilitator**: Release manager who communicates release and implementation statistics to the client and collects feedback from the end-user and client which is supplied to the coordinator, so that the changes could be made in the software immediately before the final software is released in the market.

2. Developer Side

People who are engaged in handing the technical as well as managerial issues related with the development of the software are considered under developer side. Developer side consists of analysts, designers, coders, testers, implementers, team leaders, project leaders, managers etc. Each person engaged into development handles specified task.

3. Client Side

Client side deals with the people who are usually the owners of the organisation for whom the software is being developed. They usually have different working domain but need the software to efficiently manage their task. Major generic requirements are provided by clients to the software analyst (developer end).

4. End-user Side

End users are the actual users of the system that are either purchasers of the software or are employed with the organisation for which the software is developed. The end user need varies a lot since and thus to make them satisfied with the software many custom built features have to be included. Generally they communicate with developers through clients via release managers.

B. *SDLC FEATURES*

Pipelined Stages

Each step of the project is to be completed before another is begun. Project modules are distinct and easily identifiable.

Adaptable

The model has the ability to react to operational changes as the project is developed. Change orders are easily assimilated without undue project delay and cost increases.

Repeatability

Certain aspects of the project, for example risk management, are increasingly developed as the project moves toward completion. The project expands from a prototype, in which all aspects of the project are encapsulated.

Maintenance

Systems are dynamic and the model offers to ability to produce a final project that is inherently designed for maintenance. This includes such items as cumulative documentation.

Centralised Control of Management

Management will have the ability to redirect and if necessary redefine the project once it is begun. A key phrase is 'incremental' management control, with each step under tight management control.

Quality Control

Each module of the project can be thoroughly tested before another module is begun. Project requirements are measured against actual results. Milestones and deliverables can be included for each step of the project.

Risk Management

Levels of risk are identifiable and assessment strategies available. Strategies are proved for over-all and unit risks.

Simple

The model is easy to understand and to implement. Even if the model can be expanded to various levels, a logical overview exists.

Time Management

Time-to-market considerations are central. Time-constraints are evident and a strongly structured project formulation is available.

End-User Involvement

The model lends itself to strong and constant end-user involvement. This includes project design as well as interaction during all phases of project development.

IV. **MOST COMMONLY USED TOOLS/TECHNIQUES FOR COMPARING SDLC MODELS**

To analyze a particular SDLC model and compare it with some of the existing model is only and only possible by employing any available tool, technique or method and deducing some mathematical result. This is extremely difficult and controversial as the SDLC model themselves are not mathematical models but are purely theoretical model that provides only the concept of how and what should be the methodology that must be employed for developing some specific software. The SDLC may vary for similar type of software depending on the developing organization, client organization, and available resources, expertise level of the developers and many other factors.

Say for example:

Let us assume there is software to be developed that must be able to generate payroll. Now if the organization for which the software is being developed is extremely small say about 10 employees then we can use waterfall model for developing the software. But in-case the client organization is extremely large having 100000 employees and multiple disciplines then there must be slight variation in the software being developed in accordance to the various departments. For such client the best suited SDLC will be incremental model or RAD depending upon similarities and differences among the need and requirement of the client organization.

Any employed software must have following characteristics:

1. effective
2. efficient
3. reliable
4. time saving
5. effort saving
6. cost saving
7. longer product life

C. *Merant Tracker*

Merant Tracker[10] automates the capture, management and communication of issues across project teams in development and non-development projects alike. Merant delivers the industries most flexible and comprehensive enterprise change management solutions. Already in use at thousands of organizations across the globe, Merant's products and services dramatically enhance the productivity, quality and ROI of customers' technology initiatives by allowing them to quickly and cost-effectively track, manage and control modifications in business-critical information assets.

D. *PQM Plus*

PQM Plus[11] is the Intelligent Software Measurement and Estimating Tool. It is a productivity/quality measurement system developed for software development project managers and measurement specialists. PQM *Plus* is a benchmarking and measurement tool with a robust function point repository that provides project estimating based on historical data, project scheduling, and risk assessments. PQM Plus and SMR have been designed to work together to share relevant data to aid in the production of software measurement reports. PQM Plus has received Type 1 and Type 2 certification from IFPUG, and is the only measurement tool available today that has received this level of certification. IFPUG Type 2 Certification requires PQM Plus be an "Expert system that aids in the counting of function points."

E. *SMR*

Software Measurement, Reporting and Estimating (SMR)[12] are a tool that automates software project estimating and the reporting of project performance metrics. Organizations can use SMR to estimate project size, effort, schedule and staffing early in the life-cycle using in-house and/or industry benchmarks. Once the project is complete SMR is used to capture project data, report the performance of development projects, and compare the performance to in-house and/or industry benchmarks. SMR's intuitive interface allows a user to quickly develop project estimates, enter key project statistics, compare the performance to benchmarks, analyze the results of the comparisons, and publish the report in either Word or Power Point formats. SMR has been designed to work with PQM Plus and the Function Point

WORKBENCH in order to share relevant data to aid in the production of software measurement reports.

F. *WORKBECNCH 6.0*

The Function Point WORKBENCH[12] is a network-ready Windows-based software tool which makes it easy for an organization to implement the Function Point Analysis technique for sizing, estimating, and evaluating software. The Function Point WORKBENCH is specifically designed to be sale-able for effective use by individual counters as well as for large distributed IT environments. The Function Point WORKBENCH™ and SMR have been designed to work together to share relevant data to aid in the production of software measurement reports.

G. *Putnam Model*

The Putnam model is an empirical software effort estimation model[13]. The original paper by Lawrence H. Putnam published in 1978 is seen as pioneering work in the field of software process modeling[14]. As a group, empirical models work by collecting software project data (for example, effort and size) and fitting a curve to the data. Future effort estimates are made by providing size and calculating the associated effort using the equation which fit the original data (usually with some error).

Created by Lawrence Putnam, Sr. the Putnam model describes the time and effort required to finish a software project of specified size. SLIM (Software LIfe-cycle Management) is the name given by Putnam to the proprietary suite of tools his company QSM, Inc. has developed based on his model. It is one of the earliest of these types of models developed, and is among the most widely used. Closely related software parametric models are Constructive Cost Model (CoCoMo), Parametric Review of Information for Costing and Evaluation – Software (PRICE-S), and Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM).

H. *QSM-SLIM*

SLIM-Estimate[15] helps you estimate the cost, time, and effort required to satisfy a given set of system requirements and determine the best strategy for designing and implementing your software or systems project. In addition to software cost estimation, this powerful systems and software project estimation tool provides a high level of configurability to accommodate the different design processes being used by developers today: such as Agile development, package implementation, hardware, call center development, infrastructure, model-based development, engineering and architecture design, service-oriented architecture, SAP, Oracle, and more.

I. *CoCoMo Models*

The software cost estimation model this report will focus on is the Constructive Cost Model, also known as CoCoMo. It was developed in 1981 by Barry Boehm. Boehm proposed three levels of the model; basic, intermediate, detailed. We have chosen to use the intermediate level for our cost estimation model. CoCoMo model distinguishes between three modes of software development and provides different cost and schedule equation for each mode. It produces the order-of-magnitude assessment of the expected development costs.

V. USE OF CoCoMo MODEL FOR COMPARING DIFFERENT SDLC MODELS

CoCoMo (constructive cost model) is empirical cost estimation model that is self sufficient in providing some what a clear picture in mathematical terms regarding the software being developed. CoCoMo was initially proposed by Dr. Barry W Boehm in year 1981. The model calculates the cost in terms of the effort, development time and the staffing needs.

Various other derivatives of CoCoMo 81 (as proposed by Barry Boehm) are CoCoMo-II, COSYSMO, etc.

The Constructive Cost Model (CoCoMo)[16] is an algorithmic software cost estimation model developed by Barry W. Boehm. The model uses a basic regression formula with parameters that are derived from historical project data and current project characteristics.

CoCoMo was first published in Boehm's 1981 book Software Engineering Economics[17] as a model for estimating effort, cost, and schedule for software projects. It drew on a study of 63 projects at TRW Aerospace where Boehm was Director of Software Research and Technology. The study examined projects ranging in size from 2,000 to 100,000 lines of code, and programming languages ranging from assembly to PL/I. These projects were based on the waterfall model of software development which was the prevalent software development process in 1981.

References to this model typically call it CoCoMo 81. In 1995 CoCoMo II was developed and finally published in 2000 in the book Software Cost Estimation with CoCoMo II.[18] CoCoMo II is the successor of CoCoMo 81 and is better suited for estimating modern software development projects. It provides more support for modern software development processes and an updated project database. The need for the new model came as software development technology moved from mainframe and overnight batch processing to desktop development, code re-usability and the use of off-the-shelf software components. This article refers to CoCoMo 81.

CoCoMo consists of a hierarchy of three increasingly detailed and accurate forms. The first level, Basic CoCoMo is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (Cost Drivers). Intermediate CoCoMo takes these Cost Drivers into account and Detailed CoCoMo additionally accounts for the influence of individual project phases.

A. *Basic CoCoMo*

Basic CoCoMo computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of source lines of code (SLOC)

CoCoMo applies to three classes of software projects:

- Organic projects - "small" teams with "good" experience working with "less than rigid" requirements
- Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements
- Embedded projects - developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects.(hardware, software, operational, ...)

The basic CoCoMo equations take the form

Effort Applied (E) = $a_b(\mathrm{SLOC})^{b_b}$ [man-months]

Development Time (D) = $c_b(\text{Effort Applied})^{d_b}$ [months]

People required (P) = Effort Applied / Development Time [count]

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Where, SLOC is the estimated number of delivered lines (expressed in thousands) of code for project. The coefficients a_b, b_b, c_b and d_b are given in the following table:

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Basic CoCoMo is good for quick estimate of software costs. However it does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.

B. *Intermediate CoCoMo*

Intermediate CoCoMo computes software development effort as function of program size and a set of "cost drivers" that include subjective assessment of product, hardware, personnel and project attributes. This extension considers a set of four "cost drivers", each with a number of subsidiary attributes:-

- Product attributes
  - Required software reliability
  - Size of application database
  - Complexity of the product
- Hardware attributes
  - Run-time performance constraints
  - Memory constraints
  - Volatility of the virtual machine environment
  - Required turnabout time
- Personnel attributes
  - Analyst capability
  - Software engineering capability
  - Applications experience
  - Virtual machine experience
  - Programming language experience
- Project attributes
  - Use of software tools
  - Application of software engineering methods
  - Required development schedule

Each of the 15 attributes receives a rating on a six-point scale that ranges from "very low" to "extra high" (in importance or value). An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4.
The Intermediate CoCoMo formula now takes the form:
$E=a_i(\mathrm{SLoC})^{(b_i)}.\mathrm{EAF}$
where E is the effort applied in person-months, SLoC is the estimated number of thousands of delivered lines of code for the project, and EAF is the factor calculated above. The coefficient a_i and the exponent b_i are given in the next table.

Software project	A_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

The Development time D calculation uses E in the same way as in the Basic CoCoMo.

C. *Detailed CoCoMo*

Detailed CoCoMo incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.
The detailed model uses different effort multipliers for each cost driver attribute. These Phase Sensitive effort multipliers are each to determine the amount of effort required to complete each phase.
In detailed CoCoMo, the effort is calculated as function of program size and a set of cost drivers given according to each phase of software life cycle.
A Detailed project schedule is never static.
The five phases of detailed CoCoMo are:-

- Plan and requirement.
- System design.

- Detailed design.
- Module code and test.
- Integration and test.
- COSYSMO computes effort (and cost) as a function of system functional size and adjusts it based on a number of environmental factors related to systems engineering.

COSYSMO's central cost estimating relationship or CER is of the form:

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$

where "Size" is one of four size additive size drivers, and EM represents one of fourteen multiplicative effort multipliers.

VI. RESULT AND DISCUSSION

As the proposed new SDLC Model incorporated with release management which is be known as SDLC VISHWAS. The major concept of SDLC VISHWAS states that development is an ongoing and consistent process along with the successive releases of modules of the whole software. The core concept says that the **end user (**actual person who would be using the software), **client** (the organizations who is keenly interested in purchasing software), **release managers (**the managerial team engaged in providing complete, constant and continuous communication between the end-user, via client to the developer) and the **developer** (people engaged in software development) are the major tiers of the SDLC Model. Due to the release managers the developer is constantly informed about the changing needs and requirements of the client and the development process is within the clear view of the client. Thus enabling the client to directly correlate there needs and requirements to the developers. Thus the software being developed is capable of addressing all the requirements of the end-user himself. This helps in decrease of unnecessary effort of developing software and modifying it in accordance to the need of the customer. If the same software is to be developed by say waterfall model then due to its architecture the software will be released then only the client will be able to know the limitations of the software and if changes needed then again the whole process of SDLC will have to be followed. Similarly say if the incremental model is employed then only after the first increment product is made available to the client the client can check the software for his satisfaction. If client is unsatisfied then those changes will be incorporated in the second increment product.

While due to architecture of SDLC VISHWAS the client will be immediately informed about the development and the needs or changes will be informed back to the developer via release managers. Thus repetition of SDLC phases is minimized leading to decrease in the total number of LOC (lines of codes). A comparison will be made when same software will be developed using existing SDLC and SDLC VISHWAS on the basis of LOC. To get overview of differences of SDLC VISHWAS and existing SDLC for the same software we will be employing the CoCoMo 81 which is the most simple and widely accepted cost estimation technique. Basic CoCoMo is good for quick estimate of software costs. However it does not account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and so on.

VII. CONCLUSION

The above study gives a clear understanding that various SDLC models when employed for developing different software then they may generate successful results owing to the fact that circumstances, resources, requirements, etc do vary for developer side as well as for client side. Employing a specified SDLC model for certain type software could not be determined in exact terms. Employing of any SDLC model is entirely a matter of choice which is dependent on the developer side. Thus comparing any of the SDLC models on some mathematical basis is almost impossible; they could be compared only on theoretical basis.

But if we do want to compare SDLC models mathematically then we need to develop the same software with same requirements and same developer expertise. We need to keep all the elements involved in the development of the software constant except for the SDLC model.

Out of many SDLC models the best suited is entirely dependent on developer end and client end constraints.

If we develop software using multiple SDLC model then the only method to compare SDLC models used in them with success will primarily depend upon LOC (lines of codes).

The most simple and easiest mathematical model for generating an estimation of effort, development time and staffing is CoCoMo – 81.

Employing CoCoMo – 81 will enable us to get somewhat a theoretical view displayed in mathematical form which SDLC model is more satisfactory successful for developing specified software.

ACKNOWLEDGEMENT

I express my sincere thanks to Mr. Santosh Rungta (Chairman, RCET, Bhilai), Mr. Saurabh Rungta (Director, Technical, RCET, Bhilai), Dr. Sipy Dubey (Dean – Research and Development), Prof. K. J. Satao (HOD CSE-IT-MCA) for their kind patronage and encouragement that made this paper possible. Working on this paper has been a great learning experience for me. There were moments of anxiety, when I could not solve a problem for several days and there were moments when I could solve a problem after struggling for several days, but I have enjoyed every moment of the process and are thankful to all people associated with us during this period.

REFERENCES

[1]Klopper, R., Gruner, S., & Kourie, D. (2007),0"Assessment of a framework to compare software development methodologies", *Proceedings of the 2007 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, 56-65. doi: 10.1145/1292491.1292498

[2] Software Methodologies Advantages & disadvantages of various SDLC models.mht
[3] Roger S. Pressman, Software Engineering: A Practitioner's Approach http://www.selectbs.com/analysis-and-design/what-is-the-waterfall-model
[4]Roger S. Pressman, Software Engineering: A Practitioner's Approach http://en.wikipedia.org/wiki/Incremental_build_model#Incremental_Model
[5]Boehm B, "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes", "ACM", 11(4):14-24, August 1986
[6]Roger Pressman, titled Software Engineering - a practitioner's approach
[7]Software Release Management, 6th European Software Engineering Conference, LNCS 1301, Springer, Berlin, 1997
[8]Hoek, A. van der, Wolf, A. L. (2003) Software release management for component-based software. Software—Practice & Experience. Vol. 33, Issue 1, pp. 77–98. John Wiley & Sons, Inc. New York, NY, USA.
[9]Software Release Management: Proceedings of the 6th European Software Engineering Conference, LNCS 1301, Springer, Berlin, 1997(Andre van der Hoek, Richard S. Hall, Dennis Heimbigner, and Alexander L. Wolf Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Boulder, CO 80309 USA)
[10] http://www.softmart.ru/pdf/Tracker_Release_Highlights_8.pdf
[11] http://www.qpmg.com/main_pdts.html
[12] http://www.qpmg.com/main_pdts.html#FPworkbench
[13] Putnam, Lawrence H.; Ware Myers (2003). *Five core metrics : the intelligence behind successful software management*. Dorset House Publishing. ISBN 0-932633-55-2.[14] Putnam, Lawrence H. (1978). "A General Empirical Solution to the Macro Software Sizing and Estimating Problem". IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-4, NO. 4, pp 345-361.
[15] http://www.qsm.com/tools/slim-estimate
[16] Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007 42 Software Project Planning(by narender sharma (istk))) The Constructive Cost Model (CoCoMo) Constructive Cost model (CoCoMo) Basic Intermediate Detailed Model proposed by B. W. Boehm’s through his book Software Engineering Economics in 1981.
[17] Barry Boehm. Software Engineering Economics. Englewood Cliffs, NJ:Prentice-Hall, 1981. ISBN 0-13-822122-7
[18] Barry Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. Software Cost Estimation with CoCoMo II (with CD-ROM). Englewood Cliffs, NJ:Prentice-Hall, 2000. ISBN 0-13-026692-2