OPEN ACCESS

International
Journal
Of Modern Engineering Research (IJMER)

Comparing Various SDLC Models On The Basis Of Available Methodology

Ankit Thakur[1], Deepti Singh[2], Vikas Chaudhary[3]
[1.] *M.Tech (C.S) (Scholar, Bhagwant University, Ajmer, India)*
[2.] *M.Tech (C.S) (Scholar, Bhagwant University, Ajmer, India)*
[3.] *M.Tech (C.S) (H.O.D, Bhagwant University, Ajmer, 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 model*

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 (SOFTWARE DEVLOPMENT LIFE CYCLE) 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. Phases Involved In SDLC Model

The phases that are generally present in each and every software development life cycle model are;

1. Understanding the problem (through requirements gathering).
2. Deciding a plan for a solution (Designing)
3. Coding the planned solution
4. Testing the actual program
5. Deployment & maintenance of the product.

For large systems, each activity can be extremely complex and methodologies and procedures are needed to perform it efficiently and correctly. Furthermore, each of the basic activities itself may be so large that it cannot be handled in single step and must be broken into smaller steps. For example, design of a large software system is always broken into multiple, distinct design phases, starting from a very high level design specifying only the components in the system to a detailed design where the logic of the components is specified.

The common phases of an SDLC can be represented by the following diagram.

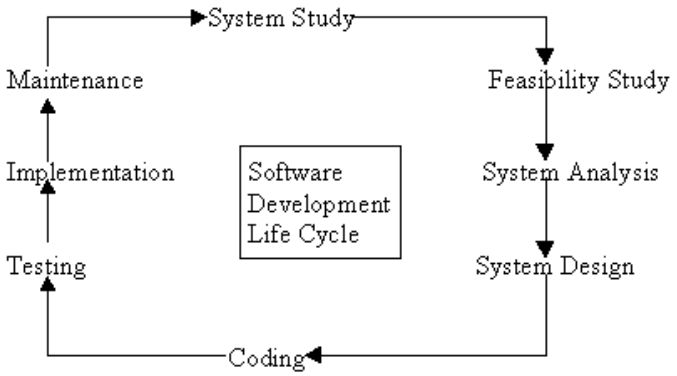

FIGURE 1.Phases of an SDLC model.

In addition to the activities performed during software development, some activities are performed after the main development is complete. There is often an installation phase, which is concerned with actually installing the system on the client's computer systems and then testing it. Maintenance is an activity that commences after the software is developed. Software needs to be maintained not because some of its Components "wear out" and need to be replaced, but because there are often some residual errors remaining in the system which must be removed later as they are discovered. Therefore, maintenance is unavoidable for software systems.

III. 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

IV. 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.

V. 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.

VI. 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.

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.

There are many SDLC models such as, Waterfall, RAD, spiral, incremental, V-shaped etc. used in various organizations depending upon the conditions prevailing there. All these different software development models have their own advantages and disadvantages. In the Software Industry, the hybrid of all these methodologies is used i.e with some modification. In this paper we have compared the different software development life cycle models on the basis of certain features like- Requirement specifications, Risk involvement, User involvement, Cost etc. on the basis of these features for a particular software project one can decide which of these software development life cycle models should be chosen for that particular project. Selecting the correct life cycle model is extremely important in a software industry as the software has to be delivered within the time deadline & should also have the desired quality. This study will make the process of selecting the SDLC model easy& hence will prove to be very effective for software industry.

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

[19] A. M. Davis, H. Bersoff, E. R. Comer, “A Strategy for Comparing Alternative Software Development Life Cycle Models”, Journal IEEE Transactions on Software Engineering ,Vol. 14, Issue 10, 1988