Evolving a New Software Development Life Cycle Model (SDLC) Incorporated with Release Management

Vishwas Massey, K. J. Satao

Abstract— Software Development Life Cycle or System development Life Cycle or simply SDLC (system and software is interchanged frequently in-accordance to application scenario) is a step by step highly structured technique employed for development of any software. SDLC allows project leaders to configure and supervise the whole development process of any software. Divide and conquer technique is widely used in SDLC models. Tasks that are complex in nature are broken down into smaller manageable components. Developers employ SDLC models for analyzing, coding, testing and deployment of software system. Efficient and effective software is developed because of SDLC model, thus making the software capable of addressing expectations of the customers, clients and the end-users. Software developed by employing the suitable SDLC models is better performers in the market when compared with their competitors. SDLC Models helps in regulating the software-system development time and helps in effective cost scheduling. In this paper we have tried to develop a model which guarantees that the development and delivery (release) teams engaged in some project have strong co-ordination and collaboration leading to enhanced productivity, efficiency, effectiveness and longer market life. This can be achieved by incorporating concept of release with basic SDLC phases (steps) with the concept of release management.

Index Terms— Release Management, Spiral Model, Software Development Lifecycle Model (SDLC), Waterfall Model.

I. INTRODUCTION

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. The ideas about the software development life cycle (SDLC) have been around for a long time and many variations exist, such as the waterfall, spiral to the new evolve model SDLC model 2010. These variations have many versions varying from those which are just guiding principles, to rigid systems of development complete with processes, paperwork, and people roles[1].

It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one. A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation. A number of system development life cycle (SDLC) models have been created: waterfall, fountain, and spiral, Build and fix, rapid prototyping, incremental, and synchronize and stabilize. Various software development life cycle models are suitable for specific project related conditions which include organization, requirements stability, risks, budget, and duration of project. One life cycle model theoretical 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]. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded prescriptive models[3]. None of these models deals with the concept of release management. Software release management is the process through which software is made available to and obtained by its users[10]. In this paper, we are dealing with different SDLC models and evolving new model with the concept of release management.

II. SDLC MODELS

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. There are tons of models, so here we discussing only few which have help us in evolving the new model.

A. Waterfall Model

The waterfall model [2,3,5,16] is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design (validation), Construction, Testing and maintenance. Small to medium database software projects are generally broken down into five stages:

Manuscript received on March 20, 2012

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

Prof. K. J. Satao, Professor & HOD, Rungta College of Engg. & Tech. Bhilai, Chhattisgarh -490024, India, (Email: kjsatao@gmail.com).

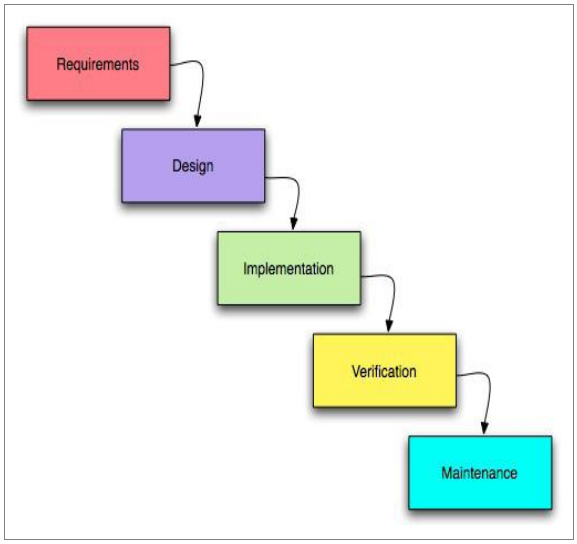

Fig 2.1: Waterfall Life Cycle Model

Fig 2.1 shows the waterfall model life Cycle phases which is the most simple, linear and sequential series of steps that is generally followed for the development of the SDLC phases. In the unmodified "waterfall model" Progress flows from the top to the bottom, like a waterfall. The waterfall development model has its origins in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware oriented model was simply adapted for software development.

Advantages

- Simple to understand and use.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.
- Each phase has specific deliverable and a review.
- Works well for projects where requirements are well understood.
- Works well when quality is more important then cost/schedule.
- Customers/End users already know about it.

Disadvantages

- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- Risk and uncertainty is high with this process model.
- Adjusting scope during the life cycle can end a project
- Not suitable for complex projects
- Users can only judge quality at the end.
- Attempt to go back 2 or more phases is very costly.
- Very risky, since one process can not start before finishing the other.

B. Prototyping Model

Often, a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach. The prototyping paradigm begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

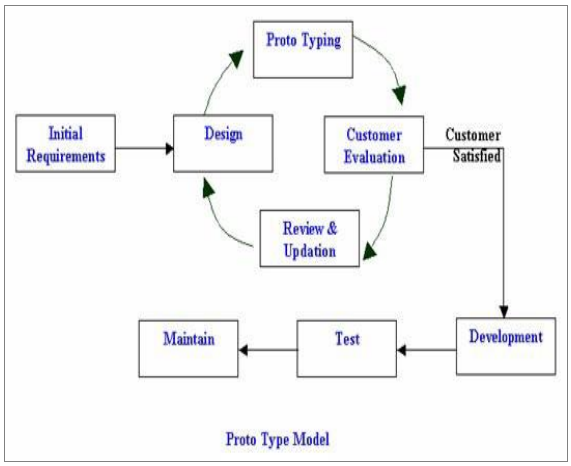

Fig 2.2 Prototyping Model

Fig 2.2 shows the different phases of Prototyping model, it is called so because in this model the prototype of the actual software system to be developed is first represented in miniature form and is then with the satisfaction and approval of the client actual software is being developed [2,12,15].

Advantages

- When prototype is shown to the user, he gets a proper clarity and 'feel' of the functionality of the software and he can suggest changes and modifications.
- Sometimes it helps to demonstrate the concept to prospective investors to get funding for project and thus gives clear view of how the software will respond.
- It reduces risk of failure, as potential risks can be identified early and mitigation steps can be taken thus effective elimination of the potential causes is possible.
- Iteration between development team and client provides a very good and conductive environment during project. Both the developer side and client side are synchronized.
- Time required to complete the project after getting final the SRS reduces, since the developer has a better idea about how he should approach the project.

Disadvantages

- Prototyping is usually done at the cost of the developer. So it should be done using minimal resources. It can be done using Rapid Application Development (RAD) tools. Sometimes the start-up cost of building the development team, focused on making the prototype is high.
- Once we get proper requirements from client after showing prototype model, it may be of no use. That is why; sometimes we refer to the prototype as "Throw-away" prototype.
- It is a slow process.
- Too much involvement of client is not always preferred by the developer.
- Too many changes can disturb the rhythm of the development team.

C. Spiral Model

The spiral model[2,14,15] 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.

Fig 2.3 shows the different phases of spiral model that is 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.

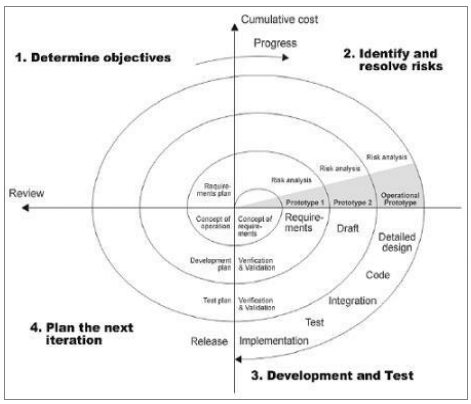

Fig 2.3: Spiral model

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.

Advantages

- Changing requirements can be accommodated.
- Allows for extensive use of prototypes
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided in to smaller parts and more risky parts can be developed earlier which helps better risk management.

Disadvantages

- Management is more complex.
- End of project may not be known early.
- Not suitable for small or low risk projects (expensive for small projects).
- Process is complex
- Spiral may go indefinitely.
- Large numbers of intermediate stages require excessive documentation.

D. SDLC 2010

This model gives the Core Control embedded and Project Management is mapped onto it. Quality Attributes such as security, safety, performance etc., are addressed well during or after the deployment of the software. Modified concept of user developer interaction (three dimensional models - comprises of the user, owner and developer). Establishes clear guideline to address: “who has to do what and when in the various stages in SDLC”. Conflicts among the developer and the user are eliminated or reduced to a manageable level, allowing timeliness of schedule. Outer circle depicts the various quality attributes pertaining to the various actions and the activities to be done during development. These quality attributes are embedded in the various activities and actions done by everyone involved in the development activities from requirement gathering to maintenance in various phases of SDLC [9,10].

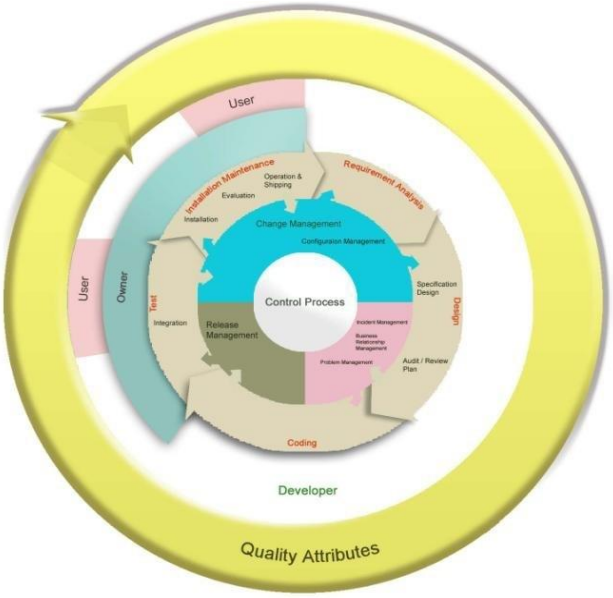

Fig 2.4 SDLC Model 2010

III. NEW PROPOSED MODEL

Model Architecture

The architecture of the model is based on the following components

· Release Manager as centralized core functional controllers for SDLC phases

· Developer side consisting of all the people responsible for technical aspects

· Client side consists of interested organization involved in purchase of software

· End user side is the actual users of the systems.

The model consists of four layers.

A. Release Manager

Release managers[10,11] (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:

1. 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. The coordinator coordinates the functional working of the other four RMs. "Coordinator" can refer to a position within an organization or business or SDLC with significant responsibilities for acting as a liaison between RMs, departments, stakeholders and information sources, which requires many non-administrative competencies.

2. Server application support engineer: RMs responsible for helping during troubleshoots problems with an application not typically at a code level but more at initial analysis level. They help in identifying the problems that might occur during the later stages of the development of the software. There might be many issues that are likely to occur and will lead to reduce functionality of the software being developed. If these futuristic problems are identified and a help desk is stood against them then the efficiency and effectiveness of the software will increase manifolds. Thus server application support engineers' responsibility increases in early stages of SDLC as most of the requirements both functional and technical are identified and most likely problems that might associate with them are identified. Following are the two main functions being performed by the server application support engineers:

a) Analyze release policy: Release policies are high-level statements of how releases are to be managed, organized, and performed in your environment. Policies include management goals, objectives, beliefs, and responsibilities. Use these topics to learn more about release policies and to learn how to view, create, and edit the policies.

- Release policy purpose: Release policies are typically written at an overall strategy level. The management directives contained in a policy determine the way in which activities and tasks within a given release work-flow are performed.
- View release policy: Use this procedure to view release policies that have been written and posted to the database. These policies provide guidelines for carrying out a release work-flow.
- Create/Edit release policy: Use this topic to specify a link to a release policy that you have created or edited. After you create the link, the policy name and description are displayed in the Release Policies table. The policy document can then be opened and reviewed by any user in your environment who is involved with IBM Tivoli Release Process Manager.

b) Analyze release plan: A release planning meeting is used to create a release plan which lays out the overall project. The release plan is then used to create iteration plan for each iteration release. It is important for technical people to make the technical decisions and business people to make the business decisions. Release planning has a set of rules that allows everyone involved with the project to make their own decisions. The rules define a method to negotiate a schedule everyone can commit to.

3. Architect: helps 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.

The main responsibilities of a software architect include:

- Limiting choices available during development by choosing a standard way of pursuing application development, creating, defining, or choosing an application framework for the application.
- Recognizing potential reuse in the organization or in the application by observing and understanding the broader system environment.
- Creating the component design having knowledge of other applications in the organization.

Software architects can also:

i. Subdivide a complex application, during the design phase, into smaller, more manageable pieces.

ii. Grasp the functions of each component within the application.

iii. Understand the interactions and dependencies among components.

iv. Communicate these concepts to developers.

Architect performs two main functions mainly:

a) Hardware-Software Designing: The most common set of requirements defined by any operating system or software application is the physical computer resources,

also known as hardware, a hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible,

and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements: Architecture, Processing power, Memory, Secondary storage, Display adapter and Peripherals. Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. Various software requirements and pre-requisites are: Platform (Operating System), APIs and drivers, Web browser, DBMS etc.

b) Build-Configure-Release: Software build refers either to the process of converting source code files into standalone software artifact(s) that can be run on a computer, or the result of doing so. One of the most important steps of a software build is the compilation process where source code files are converted into executable code. Software configuration is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines. Release candidate (RC) refers to a version with potential to be a final product, which is ready to release unless fatal bugs emerge. In this stage of product stabilization, all product features have been designed, coded and tested through one or more beta cycles with no known shows topper-class bug. A release is called code complete when the development team agrees that no entirely new source code will be added to this release. There may still be source code changes to fix defects. There may still be changes to documentation and data files, and to the code for test cases or utilities. New code may be added in a future release.

B. Gatekeeper:

It says "holds the keys" to production systems/applications and takes responsibility for their implementations. A gatekeeper is a person who controls access to something. **Gatekeeper is responsible to perform two main functions:**

a) Quality reviews are planned and documented inspections of a review item. The review item may be a product, a group of related products or part of a product. The scheduling of quality reviews is likely to represent a significant effort in terms of required resources. The earlier an error is identified the less costly are the implications and the penalty for failing to conduct adequate reviews is illustrated by the graph shown. The early detection of errors also minimizes the degree to which faults in one product can be compounded - either in its later development or in the development of related products.

b) Acceptance release is based on various tests and criteria usually created by business customers and expressed in a business domain language. These are high- level tests to test the completeness of a user story or stories 'played' during any sprint/iteration. These tests are created ideally through collaboration between business customers, business analysts, testers and developers; however the business customers (product owners) are the primary owners of these tests. As the user stories pass their acceptance criteria, the business owners can be sure of the fact that the developers are progressing in the right direction about how the application was envisaged to work and so it's essential that these tests include both business logic tests as well as UI validation elements (if need be).

C. Facilitator:

It serves as a liaison between varying business units to guarantee smooth and timely delivery of software products or updates. A facilitator is someone who helps a group of people understand their common objectives and assists them to plan to achieve them without taking a particular position in the discussion. Some facilitator tools will try to assist the group in achieving a consensus on any disagreements that preexist or emerge in the meeting so that it has a strong basis for future action. The role has been likened to that of a midwife who assists in the process of birth but is not the producer of the end result.

a) Communicate release: Rigorous planning and management of software releases and delivery. Create a Release Calendar for each type Product's Release Package to ensure that release progress is effectively communicated and planned. If a number of releases are being developed in parallel, create and communicate a high level release milestone plan. The Release Calendar should communicate both internal (development, testing) and external (deployment to live) milestones. The Release Packages and Release Calendar are more easily managed and communicated if the details are kept within a database or work flow tool.

b) Roll out Plan: Software Rollout Processes is a difficult task. It's the kind of thing that seems like it could really benefit from the collective knowledge of the IT industry.

I. Pre-deploy

A. Announcement – concentrate on what's in it for them
B. Documentation
   1. Description of what's coming
   2. Schedule of deployment
   3. Schedule of training
C. Testing in our environment w/full database
D. Roll-back plan
E. Training plan
   1. Develop trainers
   2. Pilot (same as #1?)
   3. Deployment
   4. Follow-up
F. Follow-up plan
   1. Training
   2. Support

II. Deploy

A. Pilot deploy (select group? division?)
B. Phased deploy?

III. Training

A. Brown-bag lunches
B. Webinars
C. Systems support
D. Admin support

IV. Follow-up Support

c) Verify release*:* Have we built the software right? (i.e., does it match the specification)? Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. According to the IS0 9000 standard verification is confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

d) Implementation: It is the realization of an application, or execution of a plan, idea, model, design, specification, standard, algorithm, or policy. In computer science, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Many implementations may exist for a given specification or standard. Implementation refers to post-sales process of guiding a client from purchase to use of the software or hardware that was purchased. This includes Requirements Analysis, Scope Analysis, Customizations, Systems Integrations, User Policies, User Training and Delivery. These steps are often overseen by a Project Manager using Project Management Methodologies set forth in the Project Management Body of Knowledge. Software Implementations involve several professionals that are relatively new to the knowledge based economy such as Business Analysts, Technical Analysts, Solutions Architect, and Project Managers. System implementation generally benefits from high levels of user involvement and management support. User participation in the design and operation of information systems has several positive results. First, if users are heavily involved in systems design, they move opportunities to mold the system according to their priorities and business requirements, and more opportunities to control the outcome. Second, they are more likely to react positively to the change process. Incorporating user knowledge and expertise leads to better solutions. The relationship between users and information systems specialists has traditionally been a problem area for information systems implementation efforts. Users and information systems specialists tend to have different backgrounds, interests, and priorities. This is referred to as the user-designer communications gap. These differences lead to divergent organizational loyalties, approaches to problem solving, and vocabularies.

User Concerns

- Will the system delivers the information I need for my work?
- How quickly can I access the data?
- How easily can I retrieve the data?
- How much clerical support will I need to enter data into the system?
- How will the operation of the system fit into my daily business schedule?

Designer Concerns

- How much disk storage space will the master file consume?
- How many lines of program code will it take to perform this function?
- Now can we cut down on CPU time when we run the system?
- What are the most efficient ways of storing this data?
- What database management system should we use?

Units

IV. CONCLUSION

SDLC system/software development life cycle is a step by step systematic approach from planning to testing and deployment of the software. There are some basic phases that are strictly followed in the specified order as analysis, designing, coding, testing and implementation. Different SDLC models are employed for developing variety of software's depending upon the type and usability of the software. Release Management is an entirely new concept that has come into scenario in recent days. Conceptually release management itself has some major phases that ensure smooth and timely delivery of the software-system to the client. The proposed SDLC model merges the basic phases of software development and release management in such a manner that a new SDLC model is evolved making the fullest use of release management thus increasing the effectiveness and software life in the market. The proposed model is basically a 4 – tier architecture comprising of Client side, End user, Developer side and the Release manager as individual (units) levels. It maps various release managers onto specific SDLC phases thus providing complete co-ordination of various release managers along with analyzers, designers, coders, testers over specified phases of SDLC. Release managers are provided with the centralized control over SDLC phases leading to regular release cycles. Clear guidelines have been established to address: "which RM has to do what and when in the various stages in SDLC". The above model is theoretically successful as all the phases and the roles and responsibilities of each person involved are clearly established. Their interaction and inter-relation with each other is well defined and thus this model successfully addresses all development phases that are integrated with the concept of release management.

ACKNOWLEDGMENT

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 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. Software Development Life Cycle (SDLC) – the five common principles.htm
2.] Software Methodologies Advantages & disadvantages of various SDLC models.mht
3. Craig Larman, Victor R. Basili, "Iterative and Incremental Development: A Brief History," Computer, vol. 36, no. 6, pp. 47-56, June 2003, doi:10.1109/MC.2003.1204375.
4. Curtis, Krasner, Iscoe, 1988.
5. Dr. Winston W. Royce (1929 - 1995) at www.informatik.uni-bremen.de. Retrieved 27 Oct 2008.
6. Conrad Weisert, Information Disciplines, Inc., Chicago, 8 February, 2003
7. Craig Larman, Victor R. Basili (June 2003). "Iterative and Incremental Development: A Brief History.
8. Using Both Incremental and Iterative Development.Dr. Alistair Cockburn, Humans and Technology, Crosstalk May 2008.
9. International Journal of Computer Science and Network Security, VOL.10 No.1, January 2010, Evolving A New Model (SDLC Model-2010) For Software Development Life Cycle (SDLC)PK.Ragunath, S.Velmourougan, P. Davachelvan, S.Kayalvizhi, R.Ravimohan
10. 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.
11. 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)
12. S. P. Overmyer: Revolutionary vs. Evolutionary Rapid Prototyping: Balancing Software Productivity and HCI Design Concerns. Center of Excellence in Command, Control, Communications and Intelligence (C3I), George Mason University, 4400 University Drive, Fairfax, Virginia..
13. Systems Development Lifecycle: Objectives and Requirements Bender RBT Inc.rbender@BenderRBT.com
14. Boehm B, "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes", "ACM", 11(4):14-24, August 1986
15. Roger Pressman, titled Software Engineering - a practitioner's approach.

AUTHORS PROFILE

Vishwas Massey did his B.E from C.I.T. Rajnandgaon, Chhattisgarh, India. Currently pursuing M.Tech in software Engineering from Rungta College of Engineering & Technology, Bhilai, India.

Prof. K. J. Satao is a Professor in Computer Science & Engineering at Rungta College of Engineering & Technology, Bhilai(C.G.). He has obtained his M.S. degree in Software Systems from BITS, Pilani(Rajasthan) in 1991. He has published more than 25 Papers in various reputed Journals, National & International Conferences. He is a Dean of the Computer Engineering & Information Technology in Chhattisgarh Swami Vivekanand Technical University, Bhilai. He is a member of the Executive Council and the Academic Council of the University. He is a member of CSI and ISTE since 2000. He has worked in various other Engineering Colleges for about 24 Years and has over 4 Years industrial experience as well. His research & development work is equivalent to Ph.D. degree in Computer Science & Engineering. His area of research includes Operating Systems, Editors & IDEs, Information System Design & Development, etc