Volume 3, Issue 9, September 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Evolving a new Free-Flow Software Development Life Cycle Model Integrating Concept of Kaizen

Vipul Aggarwal
C.S.E, BPIT, IP University
India

Abstract*— Software 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. In the era of software development there exist a large number of Models to develop software. Each model has its own characteristics, limitations and working environment. According to the requirements, software industry people use different models to develop different software. There are various models but none of them is capable to address the issues of client satisfaction. In this paper we develop a new model (Free-Flow SDLC) for software development that lays special emphasis on highly structured lifecycle and defining an output with each stage and also tries to fulfill the objective of the Software Engineering of developing high quality product within schedule and budget. The new proposed model is designed in such a way that it allows client and developer to interact freely with each other in order to understand and implement requirements in a better way using the concept of kaizen..

Keywords*— Software Engineering, Software Development Life cycle (SDLC), Free-Flow Model, Kaizen, Requirement Gathering, Testing

I. INTRODUCTION

Software Engineering is a discipline whose aim is the production of quality software, software that is delivered on time, within budget and that satisfies its requirements. Software Engineering is the area which is constantly growing. It is very interesting subject to learn as all the software development industry based on this specified area. There exist various models to develop software. But most of the existing Software Development Models pay less or very little attention towards client satisfaction. Client satisfaction matters. It matters not only to the client, but even more to the developer because it costs far less to retain a happy client than it does to find a new client. Satisfying client is an essential element for staying in this modern world of global competition. Client satisfaction is very necessary for the acceptance and delivery of the software product. Software projects fails in the absence of client satisfaction. Software Development Model must satisfy and even delight client with the value of software products and services. [2]

The Software Development Life Cycle (SDLC) 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 [3]. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one. A Software 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 [4].

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 [5]. None of these models deals with the concept of rigorous requirement gathering management and robust testing techniques. **Requirement gathering** is the process through which requirements of the software to be built is gathered and obtained by its users and **Testing** is the technique wherein the developed software is tested through different phases to check the stability and robustness of the software [10]. In this paper, we are dealing with and augmenting different phases of Software Development Lifecycle Model (SDLC) which helps us in providing an efficient, reliable, easy to implement and effective during and after implementation.

This paper is structured as follows: Section 2 throws the light on the details of various Software Development Life Cycle (SDLC) models proposed in the earlier papers. Section 3 provides an insight to the details of the existing Software Development Life Cycle (SDLC) models and the proposed model in this paper using modified Requirement gathering

phase and augmented testing phase. Section 4 describes the details of how new features are integrated and executed in our paper. Section 5 ends the paper with a conclusion.

II. RELATED WORK

Plenty of work has been done for software development and testing. Most of the studies focus on improving the development process by using various testing techniques. Each change made to a piece of software, each new piece of functionality, each attempt to fix a defect, introduces the possibility of error. With each error, the risk that the software will not fulfill its intended purpose increases. The only thing we can do to reduce the number of errors already present is to test it. By following a cycle of testing and rectification we can identify issues and resolve them. [6]

Kirti et.al, International Journal of Latest Research in Science and Technology proposes V-Model is the best model for development as it is very easy to use and understood and each phase has some specific deliveries and less chances of downward flow of defects. [7]

Starting from the requirements, the system is developed one phase at a time until the lowest phase, the implementation phase, is finished. At this stage testing begins, starting from unit testing and moving up one test level at a time until the acceptance testing phase is completed. Splitting the testing into V-Model test levels that are performed in parallel could mean that the testing process is easily manageable and more flexible than traditional VModel testing. [8]

Vijay.N proposed a new technique to improve the development process by using effective testing technique by focusing on time constraints and suggests that if the time span from the Requirement Identification to the release is minimized, would make the software more profitable. For using V-Model as software development standard he structured it into sub models, each sub model performs some specific activities. These sub models are Project management (PM).

When evidence exists that the resulting system meets requirements, the system and its components are accepted. The V-Model integrates planning and execution of testing throughout the life cycle. During requirements elicitation and analysis, acceptance test planning is initiated, including development of acceptance test scenarios. The plans and scenarios for acceptance testing are executed prior to deploying the system. During design activities, system and integration test planning and scenarios are initiated. These test plans and scenarios are executed after units of software are coded and unit tested. (30 May, 2008). [10] [11]

[12] Proposed a simulation planning that must be completed prior to starting any development process. Its purpose is to identify the structure of the project development plan and to classify what must be simulated, the degree of simulation, and how to use the simulation results for future planning. Moreover, the approach takes into consideration such issues as configuration requirements, design constraints, development criteria, problem reporting and resolution, and analysis of input and output data sets.

[13] Described three types of simulation methodologies. The first is called “simulation as software engineering” and revolves around simulating the delivery of a product. This comprises the use of large simulation models to represent a real system at the production environment. The second is called “simulation as a process of organizational change” and revolves around the delivery of a service. This comprises the use of temporary small-scale models to simulate small-scale tasks and processes. The third is called “simulation as facilitation” and revolves around understanding and debating about a problem situation. This comprises using “quick-and-dirty” very small-scale models to simulate minute-by-minute processes.

[14] Proposed the use of simulation as facilitation based on system dynamics. The model proposes the simulation of three development stages: The conceptualization stage which simulates problem situation and system objectives; the development stage which simulates the coding, verification, validation, and calibration processes; and the facilitation stage which simulates group learning around the model, project findings, and project recommendations.

[15] Proposed a guideline to be followed for performing a simulation study for software development life cycles. It is composed of ten processes, ten phases, and thirteen reliability evaluation stages. Its purpose is to assess the credibility of every stage after simulation and match it with the initial requirements and specifications. The model provides one of the most documented descriptions for simulating life-cycles in the software engineering field [13].

[16] Proposed a software engineering process simulation model called SEPS for the dynamic simulation of software development life cycles. It is based on using feedback principles of system dynamics to simulate communications and interactions among the different SDLC phases and activities from a dynamic systems perspective. Basically, SEPS is a planning tool meant to improve the decision-making of managers in controlling the projects outcome in terms of cost, time, and functionalities.

[17] Proposed a discrete open source event simulation model for simulating the programming and the testing stages of a software development process using MathLab. The model investigates the results of adopting different tactics for coding and testing a new software system. It is oriented toward pair programming in which a programmer writes the code and the simulation acts as an observer which reviews the code and return feedback to the original programmer.

[18] Proposed an intelligent computerized tool for simulating the different phases of a generic SDLC. It is intended to help managers and project directors in better planning, managing, and controlling the development process of medium-scale software projects.

III. FRAMEWORK

A. INTIAL FRAMEWORK

Software process models are defined only in terms of requirements analysis phase of each model.

1) *WATERFALL MODEL:* The requirement gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the

information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer [20].

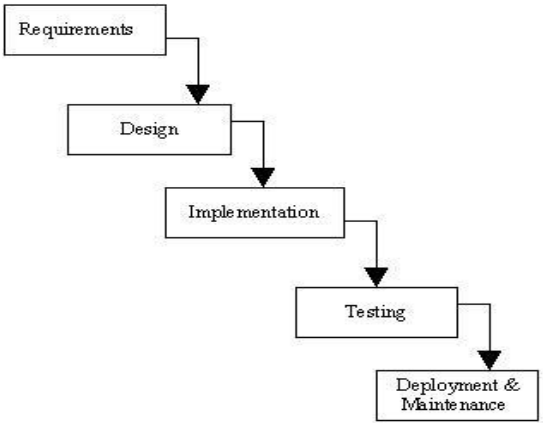

Figure 1: Waterfall Model [19]

The major weakness of the Waterfall Model, as in figure 1, is that after project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements change or more information becomes available to the project team because requirements almost always change during long development cycles, often the product that is implemented at the end of the process is obsolete as it goes into production. The Waterfall Model is a poor choice for software development projects where requirements are not well-known or understood by the development team. It might not a good model for complex project or projects that take more than a few months to complete. [22]

2) SPIRAL MODEL

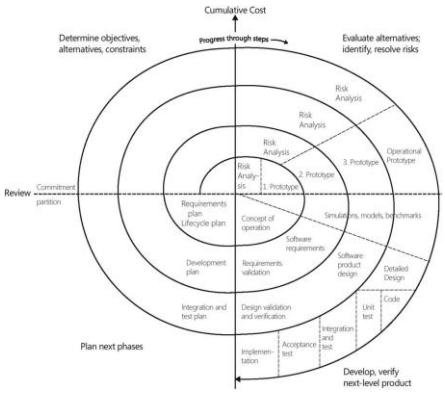

Figure 2: Spiral Model [20]

In response to the weaknesses and failures of the Waterfall SDLC Model, many new models were developed that add some form of iteration to the software development process. In the Spiral SDLC Model as in figure 2 , the development team starts with a small set of requirements and goes through each development phase (except Installation and Maintenance) for those set of requirements [8]. Based on lesson learned from the initial iteration, the development team adds functionality for additional requirements in ever-increasing "spirals" until the application is ready for the Installation and Maintenance phase [20][21]. [22]

3) INCREMENTAL MODEL

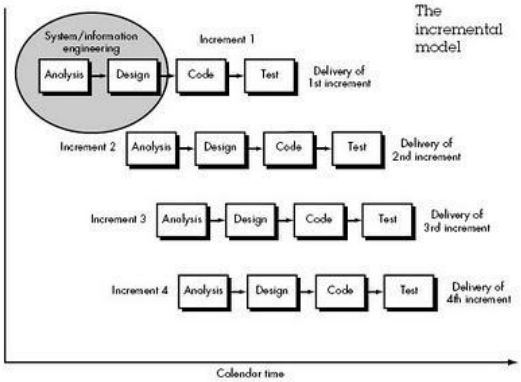

Figure 4: Incremental Model [20]

The incremental model figure 4 combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software when an incremental model is used; the first increment is often a core product [20]. That is basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality [21]. [22]

B. PROPOSED FRAMEWORK

The new proposed framework is developed by incorporating the concept of kaizen with the Free Flow Model. In the proposed framework of **Free Flow software methodology,** the development process of a software is augmented at every stage by incorporating different matrices and allowing free flow of control in the software development hierarchy in order to save time in situations where the testing team need to interact with the another team. The proposed model allows the testing team to directly contact a team without involving the intermediate teams and thereby saving time and increasing efficiency of the software development process.

In Free Flow SDLC**,** the **concept of Kaizen** also has been integrated which emphasis on the concept of continuous improvement throughout the lifecycle of software development and during its maintenance. It divides the time duration and phases of SDLC among different persons involved in the hierarchy. Kaizen helps to improve the efficiency of the product being developed by laying emphasis on the quality iteratively. Concept of kaizen has been shown in figure 5 as below:

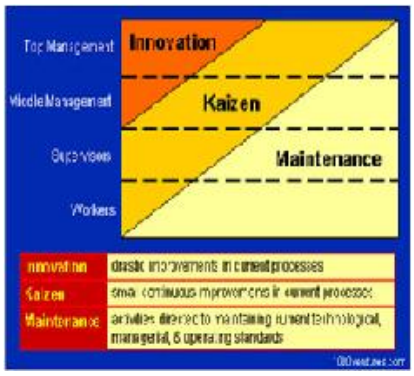

Figure 5: Concept of Kaizen

1) MODEL ACHITECTURE OF FREE-FLOW MODEL

The architecture of the model is based on the organized phases of SDLC and the free flow of control between these phases. The up and down arrows denote the flow of control during software development in any direction i.e. upwards or downwards.

With each phase a definite output matrix is associated which helps to keep track on the development activities and modify them according to the requirement of the project. **Free-Flow Model** eliminates the risks and errors in early stages through pipelined flow of processes and control. Complete architecture of Free-Flow Model is shown as below in Figure 6.

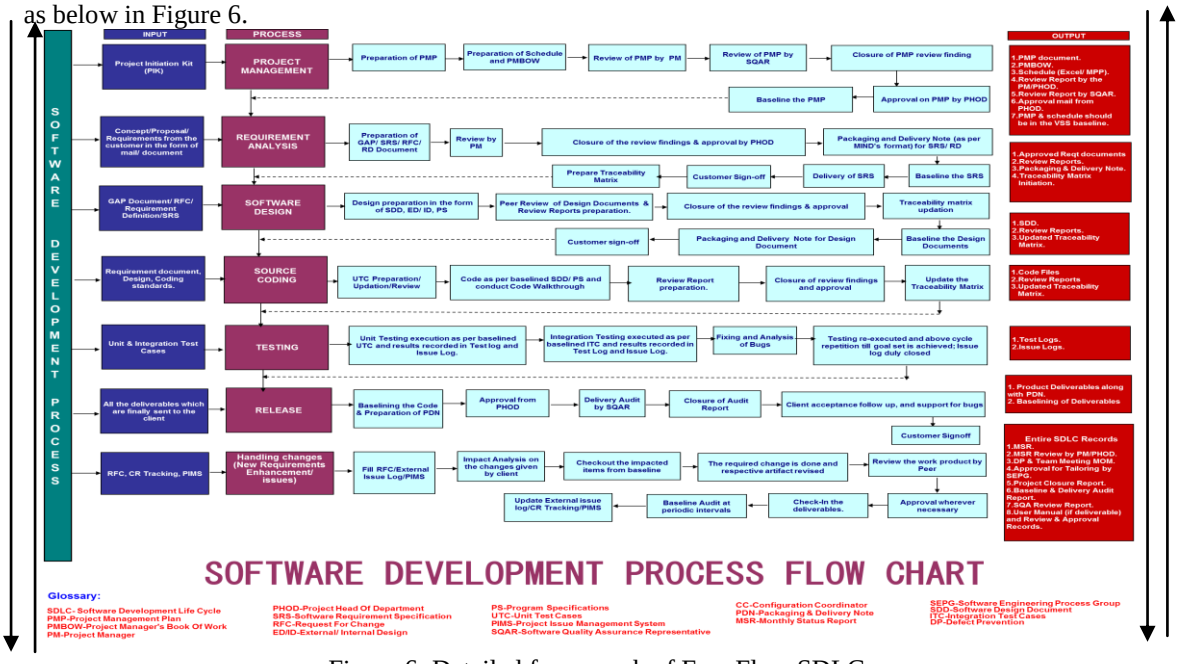

Figure 6: Detailed framework of Free Flow SDLC

Figure 6 also indicates the inputs from customer at every stage to the respective managers associated with corresponding phases to include the changes and modifications according to his requirements. It helps customer and the organisation to interact freely while winning customer's trust at every phase.

Also, in existing SDLC's main drawback was insufficient number of matrices at different stages of development lifecycle. In Free-Flow Lifecycle Model besides allowing the teams to communicate freely and navigate in any direction, a number of matrices have been added at each stage to make the system robust and increase the efficiency of the product (output) at each stage. Below matrices have been listed at each stage which have been added apart from the existing matrices.

- *Requirement Gathering, Documenting and Design Phase*
  - Process Overview Matrix
  - Process Complexity Matrix
  - FTE Analysis Matrix
  - Current Technology Matrix
  - Historical Data Matrix
  - TimeLine Matrix
  - Scope of Processes Matrix
  - Project Review Methodology Matrix
  - Content Process Flow Matrix
  - Identifying Critical and Non-Critical Activities Matrix
- *Development and Deployment Phase*
  - Process Roadmap Matrix
  - Communication Plan Matrix
  - Governance Structure Matrix
  - Knowledge Transfer Matrix
  - Technology Set-Up Matrix
  - Target Timelines Matrix
  - Performance review Matrix
  - Pilot Operations Matrix
  - Prioritization of Test Case Matrix
  - Test Case Performance Matrix
- *Steady State and Maintenance Phase*
  - 100% volume Takeover Matrix
  - Adherence to agreed SLAs
  - Process performance Reviews
  - Response to queries Matrix
  - Post Delivery support Matrix
  - Behaviour Matrix

2) FREE-FLOW SDLC FEATURES

Requirement Capture

We capture requirements through conference calls, interfacing with on-site resources, client side visits and studying existing code and implementations.

Prototype and High Level Diagrams

Documents are always necessary in a well-defined process but it is very difficult for customers to figure out upfront what they will get for their money. Prototype and High level diagrams help the clients to have a clue of what will be delivered after coding.

Module releases

Every project is divided into multiple modules. As soon as a module is completed we send the demonstration version to the customer. This enables customers to track progress and notify us with any change in flow.

Check-List Method

From planning to development, drastically reducing the commonly made mistakes by developers while coding, releasing and deploying.

Well-Defined Architecture

The above architecture helps us to train newly recruited developers and leaders quickly. The responsibilities are clearly defined at every stage which helps in improving the efficiency during the development.

Task and bug tracking system

Tasks while development and bugs while debugging are the most important aspects of the projects.. Generally it becomes very difficult to manage bugs where multiple Quality Engineers and developers are working on an application. For this, we have a project management application to manage/track both bugs and tasks .Use of above model with advanced reporting, tracking and auditing features not only makes it easy to manage but also enables improvements in each project.

IV. PERFORMANCE EVALUATION

Performance evaluation is the key aspect of undertaking any research work. So I have evaluated the work and finally concluded by elaborating the efficiency of our work. But before discussing our execution, I first present an overview of the work done and then its relevant efficiency on how it leads an organisation to produce better outputs in comparison to other existing models.

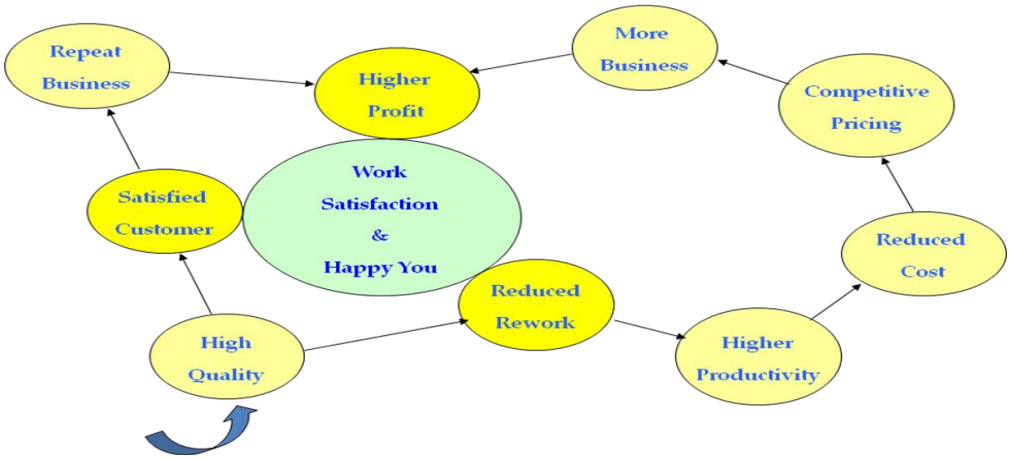

Figure 6: Benefits of Free-Flow SDLC

Figure 6 throws light on the non-tangible benefits of the free flow model for an organisation. Free Flow model is an evolutionary method whose benefits can be seen tangibly and non-tangible. It will help different organisations to develop projects effectively. It also holds its importance in every scale of projects ranging from small-scale projects to large scale projects.

FEATURES	WATERFALL	SPIRAL	INCREMENTAL	FREE-FLOW MODEL
UNDERSTANDING REQUIREMENTS	WELL UNDERSTOOD AT BEGINNING	NOT WELL UNDERSTOOD AT BEGINNING	WELL UNDERSTOOD AT BEGINNING	WELL UNDERSTOOD AT BEGINNING
COST	LOW	HIGH	MEDIUM	LOW
SCHEDULE	WITHIN SCHEDULE	SCHEDULE MAY EXCEED	SCHEDULE MAY EXCEED	WITHIN SCHEDULE
RISK INVOLVEMENT	HIGH	LOW	MEDIUM	VERY LOW
USER INVOLVEMENT	LOW	HIGH	HIGH	HIGH
GUARANTY OF SUCCESS	LOW	GOOD	HIGH	HIGH
CLIENT SATISFACTION	LOW	HIGH	HIGH	HIGH
FLEXIBILITY	RIGID	FLEXIBLE	FLEXIBLE	FLEXIBLE
TIME FRAME	MEDIUM	SHORT	VERY LONG	SHORT
INITIAL PRODUCT FEEL	NO	YES	NO	YES

Figure 7: Tabular comparison of Free-Flow Model with other Models

Figure 7 provides us with a comparison between different Software development models. The comparison highlights free flow model more agile, adaptable, error-free and robust in comparison to other SDLC models.

V. 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. The proposed work can be summarized as the creation of the approach FREE-FLOW SDLC to develop software more efficiently. The aim of Software Engineering is to develop software of high quality within budget and schedule. The proposed plan tries to fulfil the objective of Software Engineering by defining a well-defined procedure and targets for the client to discover the requirements efficiently from the client in order to estimate cost, schedule and effort more accurately and map them further in the development lifecycle. With the proposed work, the effectiveness of optimization has been studied carefully. Further investigation to the topic reveals that FREE-FLOW SDLC can give good results. The concept has been worked out and can be used in future.

ACKNOWLEDGMENT

I take this opportunity to express our sincere thanks and deep gratitude to all those who extended their wholehearted cooperation and have helped me in completing this work successfully. I express our sincere thanks to Mr. Suyash Gupta (DTU) for his encouragement and valuable suggestions.

REFERENCES

[1] K. K. Aggarwal, Yogesh Singh Software Engineering 3rd Edition.
[2] Naresh Kumar, A. S. Zadgaonkar, Abhinav Shukla, Evolving a New Software Development Life Cycle Model SDLC-2013 with Client Satisfaction, *International Journal of Soft Computing and Engineering (IJSCE)*, , Volume-3, Issue-1, March 2013
[3] Software Development Life Cycle (SDLC) – the five common principles.htm
[4] Software Methodologies Advantages & disadvantages of various SDLC models.mht
[5] 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.
[6] Nick Jenkins,"A Software Testing Primer",An Introduction to Software Testing,2008
[7] Raymond Lewallen,"Software Development Life Cycle",2005 [4] Hee-Gyun Yeom, and Sun-Myung Hwang, "A Study on Tool for supporting the Software Process Improvement*" International Journal of Software Engineering and Its Applications* Vol.3, No.2, April, 2009
[8] Jakobsson,"V-Model Testing –Process model configuration using SVG", PMoC 14/04/2003 Version 1.5
[9] Brian Marick,"New models for test development" *Reliable Software Technologies,*1999. Version 1.0 of 03/28/00
[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] Kriti Nagpal, Raman Chawla, *International Journal of Latest Research in Science and Technology,* Vol.1,Issue 3 :Page No.217-224 ,September-October (2012).
[12] Jim Ledin, "Simulation Planning" PE, Ledin Engineering, 2000
[13] Robinson, S., "Modes of simulation practice: approaches to business and military simulation", *Proceedings in Simulation Modeling Practice and Theory,* vol. 10, pp. 513-523, 2002.
[14] Robinson, S., "Soft with a hard centre: discrete-event simulation in facilitation", *Journal of the Operational Research Society,* vol. 52, pp. 905-915 , 2001.
[15] Balci, O., "Guidelines for successful simulation studies", *Proceedings of the Simulation Conference*, pp. 25-32, New Orleans, LA, 1990.
[16] R. Sargent, R. Nance, C. Overstreet, S. Robinson, and J. Talbot, "The simulation project life-cycle: models and realities", *Proceedings of the Winter Simulation Conference, 2006.*
[17] Chi Y Lin, Tarek Abdel-Hamid, and Joseph S Sherif, "Software-Engineering Process Simulation model (SEPS)", *Journal of Systems and Software*, Vol. 38, no. 3, pp. 263-277, 1997
[18] Shmuel Ur, Elad Yom-Tov and Paul Wernick, *An Open Source Simulation Model of Software Development and Testing*, *Hardware and Software, Verification and Testing*, Lecture Notes in Computer Science, Springer, vol. 4383, pp. 124-137, 2007.
[19] Ian Sommerville, " Software Engineering", 8th Edition, 2006, pp. 89.
[20] R.S. Pressman, "*Software Engineering, A Practitioner's Approach*", 5th ed. New York: McGraw-Hill, 2001, pp. 26.
[21] B.W. Boehm, "*A Spiral Model for Software Development and Enhancement*", IEEE, IEEE Computer Society, vol. 21, issue 5, May 1988, pp. 61 – 72.
[22] A Survey of project scenario impact in SDLC models selection process, Manish Sharma

