Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies

Gaurav Kumar
Deptt. of Computer Science & Engineering,
G. J. U. S. & T., Hisar, Haryana, India
er.gkgupta@gmail.com

Pradeep Kumar Bhatia
Deptt. of Computer Science & Engineering,
G. J. U. S. & T., Hisar, Haryana, India
pkbhatia.gju@gmail.com

***Abstract*— Software Engineering aims to produce a quality software product that is delivered on time, within the allocated budget, and with the requirements expected by the customer but unfortunately maximum of the times this goal is rarely achieved. A software life cycle is the series of identifiable stages that a software product undergoes during its lifetime. However, a properly managed project in a matured software engineering environment can consistently achieve this goal. This research is concerned with the methodologies that examine the life cycle of software through the development models, which are known as software development life cycle. Hereby, we are representing traditional i.e. waterfall, Iteration, Spiral models as well as modern development methodologies like Agile methodologies that includes Extreme programming, Scrum, Feature Driven Development; Component based software development methodologies etc. All of these models have advantages and disadvantages as well. Therefore, the main objective of this research is to represent different models of software development by showing the good and bad practices of each model. A comparative analysis of traditional as well as modern methodologies is made.**

Keywords—Software Engineering, Software Engineering Models, Traditional models, Modern models.

I. INTRODUCTION

The goal of Software Engineering is to provide models and processes that lead to the production of well-documented maintainable software in a manner that is predictable. "The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase". Software has been developed from a tool used for analyzing information or solving a problem to a product in itself. However, the early programming stages have created a number of problems turning software an obstacle to software development particularly those relying on computers. Software consists of documents and programs that contain a collection that has been established to be a part of software engineering procedures. Moreover, the aim of software engineering is to create a suitable work that constructs programs of high quality. Different lifecycle model may map the basic development activities to phase in different way, thus no matter which lifecycle model is followed. The basic activities are included in all lifecycle models that may be carried out in different order in different lifecycle models. Many type of Software engineering models are available from last decades and currently also more advancement is being done in the software engineering models. Traditional software development processes are not much efficient to manage the rapid change in requirements. Traditional SDLC methods are Predictive rather than Adaptive. They want the answers up front; have a hard time to deal with change. They rely on Documentation rather than Trust. They focus on Roles rather than Teamwork and also they emphasize planning over building. Traditional models like Water fall model is easy to understand and reinforces the notion of "define before design" and "design before code". The model expects complete & accurate requirements early in the process, which are unrealistic, while Modern Software Engineering models like Agile Methodologies are a group of software development methods that are based on iterative and incremental development. The four major characteristics that are fundamental to all agile methodologies are: adaptive planning, iterative & evolutionary development, rapid and flexible response to change and promote communication. Agile methodologies are used to achieve higher quality software in a shorter period of time, self organizing teams, customer collaboration, less documentation and reduced time to market. Agile methodology includes a family of lightweight methods that include Scrum, Crystal Clear, Extreme Programming (XP), Adaptive Software Development (ASD), Feature Driven Development (FDD), and Dynamic Systems Development Method (DSDM) Crystal, Lean Software Development etc.

II. SOFTWARE ENGINEERING MODELS

Software Engineering models can be categorized into 2 categories as shown in figure 1. Parameters that affect a software engineering project are:

- Active Stakeholder Participation
- Truly meeting customer's requirement
- High degree of customer involvement
- Periodic product showcase
- Quickly responding to customer
- Effective measurement of software requirements
- Measurement of development team's velocity
- Matching between requirements and velocity
- Coordination of working plans for different roles
- Quality of delivered system
- Cost of transferring and end user training
- Product delivery cycle

978-1-4799-4910-6/14 $31.00 © 2014 IEEE
DOI 10.1109/ACCT.2014.73

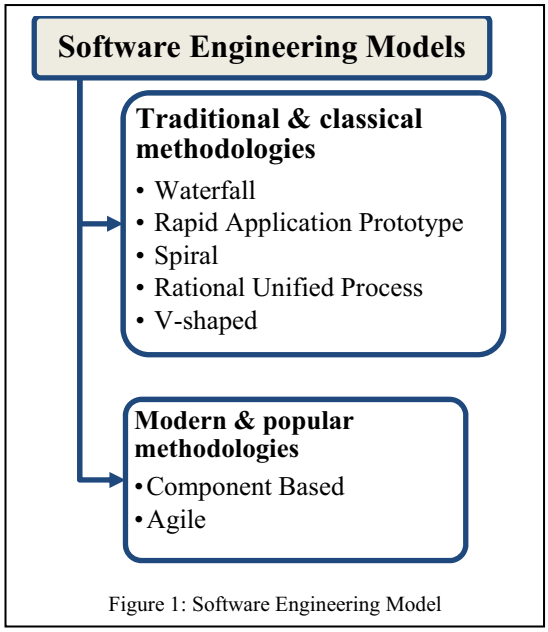

Figure 1: Software Engineering Model

- Flexibility of information system development
- Purchase and Supplier Relationship Management
- Self Organizing Teams [Jeremy, 8].
- Reduced Documentation [Pekka et. al. 2002].
- Responding to Change
- Competent and experienced team
- Flexible Design
- Trainings of professionals [Emilio, 9].
- Refactoring of code

III. Traditional & Classical Software Engineering Methodologies

The software engineering tasks include analysis of the system requirements, Design, development, implementation of the software, and testing the software to verify that it satisfies the specified requirements [Vinay Tiwari, 2011]. Hereby in the next subsections, we are going to illustrate some traditional models.

A. Waterfall Model

This model was developed by Royce in 1970 and is called "waterfall model" because its pictorial representation looks like a cascade of waterfalls as shown in figure 2. This model is easy to understand and implement; and one of the oldest models that is widely used in government projects and in many major companies. The waterfall model serves as a baseline for many other lifecycle models. This model reinforces good notion of define-before- design, design-before-code [Sommerville, 2004].

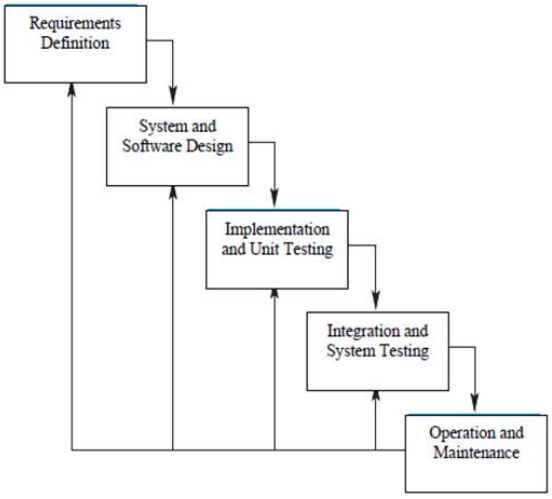

Figure 2: Waterfall Model

Although waterfall model is instructive, it has some shortcoming also. Some of these are:

- It is difficult to define all requirements at the beginning of a project
- This model is not suitable for accommodating any change
- A working version of the system is not seen until late in the project's life
- It does not scale up well to large projects.
- Real projects are rarely sequential.
- Difficult to integrate risk management.

B. Rapid Application Development (RAD) Model

This model was developed by IBM in 1980 and is shown in figure 3. In this model a rapid prototype is built up and given to user for evaluation & obtaining feedback. Then on the basis of user feedback Prototype is refined.

Figure 3: RAD Model

This model is not appropriate in the absence of user participation. Also reusable components are required to reduce development time. Highly specialized & skilled developers are required and such developers are not easily available [K.K. Agarwal, 2007].

C. Spiral Model

This model was developed by Barry Boehm in 1986 that includes project risk factor and is shown in figure 4. The radial dimension of the model represents the cumulative costs. Each path around the spiral is indicative of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360◦ represents one phase. One phase is split roughly into four sectors of major activities [Nabil, 2010].

- Objective Setting: Determination of objectives, alternatives & constraints.

- Risk Analysis: Analyze alternatives and attempts to identify and resolve the risks involved.
- Development: Product development and testing product.
- Planning: The project is reviewed and the next phase of the spiral is planned.

An important feature of the spiral model is that each phase is completed with a review by the people concerned with the project (designers and programmers). Wide range of options is available to accommodate the good features of other life cycle models. It becomes equivalent to another life cycle model in appropriate situations. High amount of risk analysis can be made because of which this model can be deployed for large and mission-critical projects.

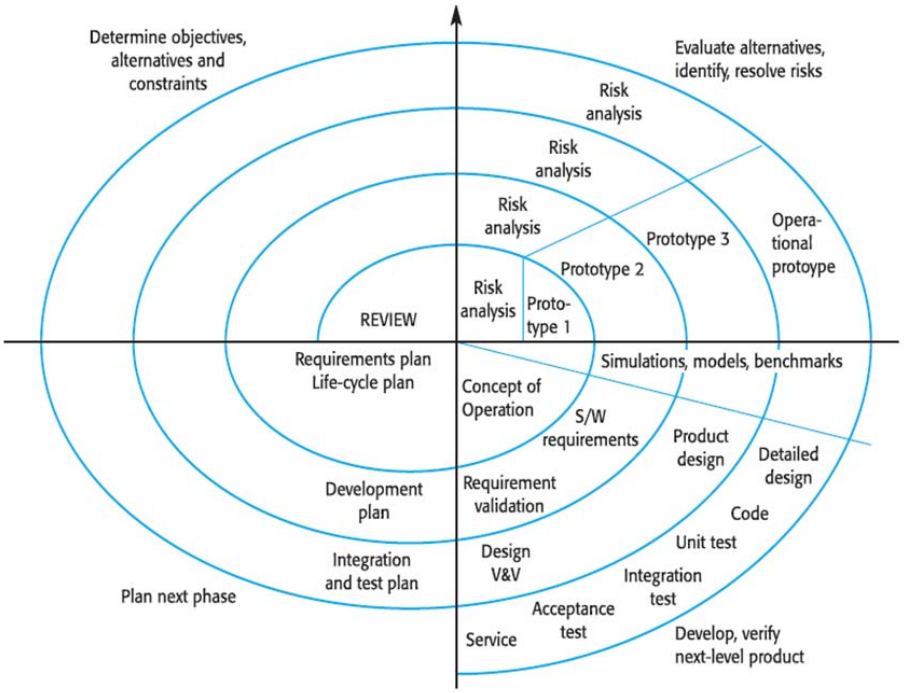

Figure 4: Spiral Model

Despite of these merits, it has shortcomings also, some of these are:

- lack of explicit process guidance in determining objectives, constraints, alternatives;
- doesn't work well for smaller projects
- relying on risk assessment expertise;
- can be a costly model to use.
- provides more flexibility than required for many applications.

D. Rational Unified Process (RUP) model

This model is developed by I. Jacobson, G. Booch and J. Rumbaugh; and Maintained and enhanced by Rational Software Corporation. RUP is an iterative approach for object oriented systems. At a high level, the unified process is organized around two concepts: phases and workflows as shown in figure 5 and 6 [K.K. Aggarwal et. al., 2007].

Figure 5: Phases of RUP Model

- *Inception* defines scope of the project so that needs of every stakeholder are considered.
- In the *Elaboration phase*, project plan is defined to determine resources, architecture etc. are most suitable for the project. At the end of Elaboration phase, an analysis is made to determine the realization of risks, stability of vision of what the product is to become, the stability of the architecture, and the expenditure of resources vs. what was initially planned.
- In the C*onstruction phase*, objectives are translated into design and architecture documents.

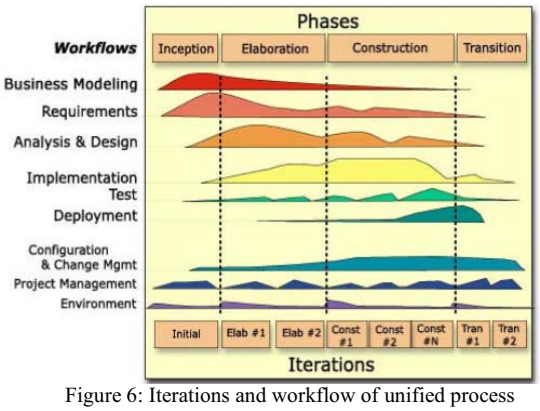

Figure 6: Iterations and workflow of unified process

- *Transition phase* includes delivering, training, supporting and maintaining the product.

Throughout the phases, nine workflows take place in parallel. These are: Business Modeling, Requirements, Analysis & Design, Implementation, Test, Configuration & Change Management, Project Management and Environment.

E. V-Shaped Model

The V-model may be considered as an extension of the waterfall model as shown in figure 7. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-model establishes the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction respectively. V- Model is simple and easy to use. Each phase has specific deliverables. Higher chance of success over the waterfall model due to the early development of test plans during the life cycle. Works well for small projects where requirements are easily understood [Nabil, 2010].

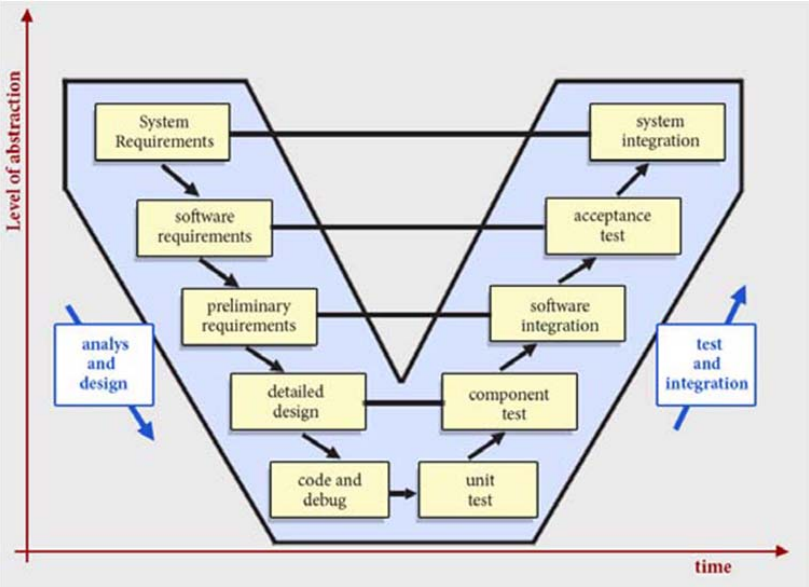

Figure 7: V-shaped model

Some of the shortcomings of V-model are:

- Very rigid like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- This Model does not provide a clear path for problems found during testing phases.

IV. MODERN SOFTWARE ENGINEERING METHODOLOGIES

A. Component Based Software Development Methodology

Component Based Software Development (CBSD) is focused on assembling existing components to build a software system, with a potential benefit of delivering quality systems by using quality components. It departs from the conventional software development process in that it is integration centric as opposed to development centric. The quality of a component based system using high quality components does not therefore necessarily guarantee a system of high quality, but depends on the quality of its components, and a framework and integration process used. Hence, techniques and methods for quality assurance and assessment of a component based system would be different from those of the traditional software engineering methodology [Mostefai et. al., 2011].

Since components are reused in several occasions, they are likely to be more reliable than software developed from scratch, as they were tested under a larger variety of conditions. Cost and time savings result from the effort that would otherwise be necessary to develop and integrate the functionalities provided by the components in each new software application. The component model highlighting the elements that make up the components model, support for development, communication, evolution, composition and implementation of components etc. are shown in figure 8.[María et. al., 2009]

Unlike the traditional metrics, they do not depend on the component's code size, which is generally unknown. Component performance and reliability also vary because component level testing may be limited to black box testing [S. Mahmood et. al, 2005].

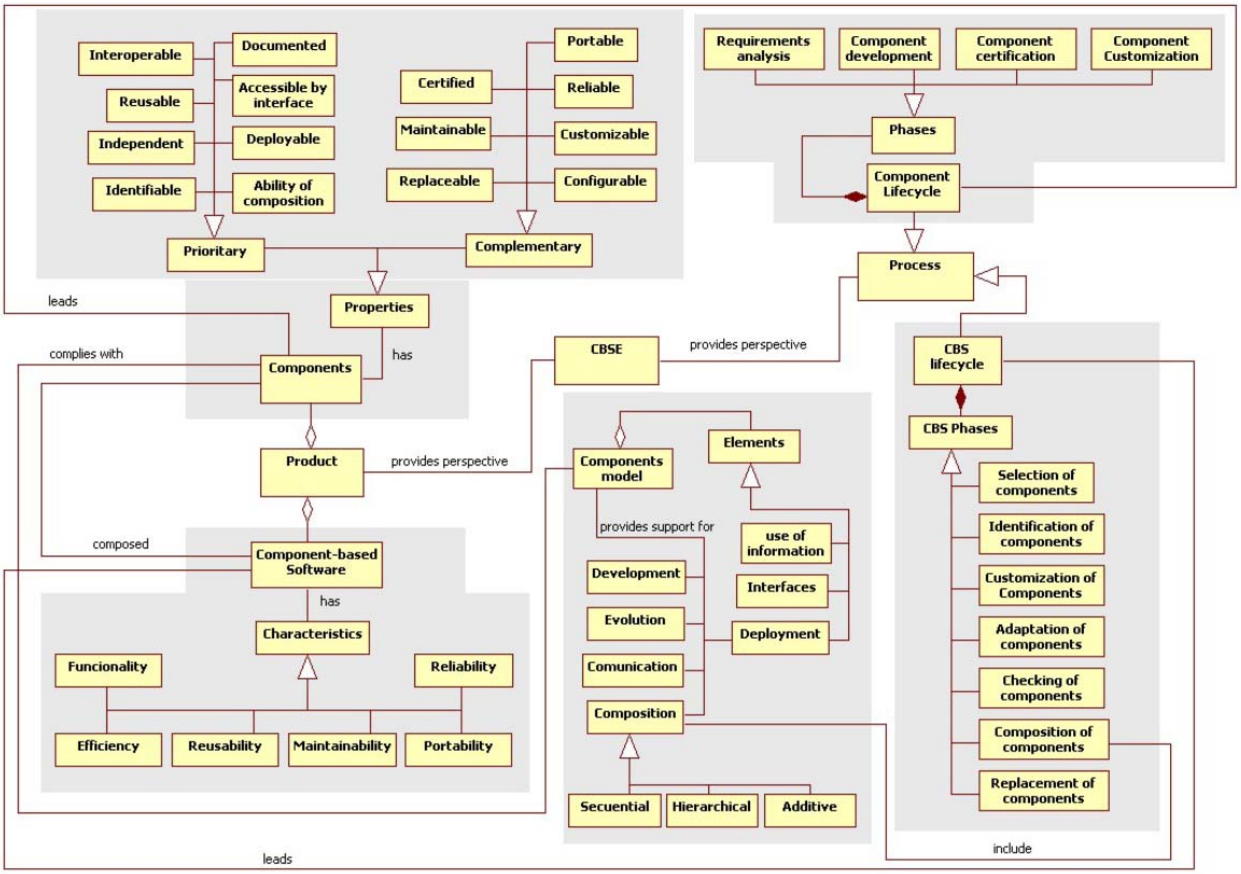

Figure 8: Integrated conceptual model for CBSE

B. Agile Methodology

Agile methodology is an approach to project management, typically used in software development. It helps teams respond to the unpredictability of building software through incremental, iterative work strategy, known as sprints. The twelve principles behind agile manifesto are:

- The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile methodology includes several models. Some of the popular agile methodologies are:

1) Extreme Programming model

In this Developers are responsible for testing their own code. Extreme Programming is designed for development teams of between 3 and 10 engineers. The team is enhanced with one of more customer representatives, and managers to provide architectural input and to act as coding mentors or *coaches*. [Michael J Rees, 2002]

"Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situations"[Jeffries, Ron 2001][Nayan,

2006]. The life cycle of XP consists of 5 phases [Pekka et. al. 2002]:

- In the *Exploration phase*, customer describes the story cards in the form of features that has to be included in the first release.
- In the *Planning phase*, priority is set for the user stories, estimation of effort and schedule for each story is made.
- In the *Iterations to release phase*, schedule set in the planning stage is broken down into a number of iterations that will each take one to four weeks to implement. At the end of last iteration, the system is ready for production.
- In the *Productionization phase*, new changes may still be found and the decision has to be made if they are included in the current release.
- In the *Death phase*, necessary documentation of the system is finally written as no more changes to the architecture, design or code are made. Death may also occur if the system is not delivering the desired outcomes, or if it becomes too expensive for further development.

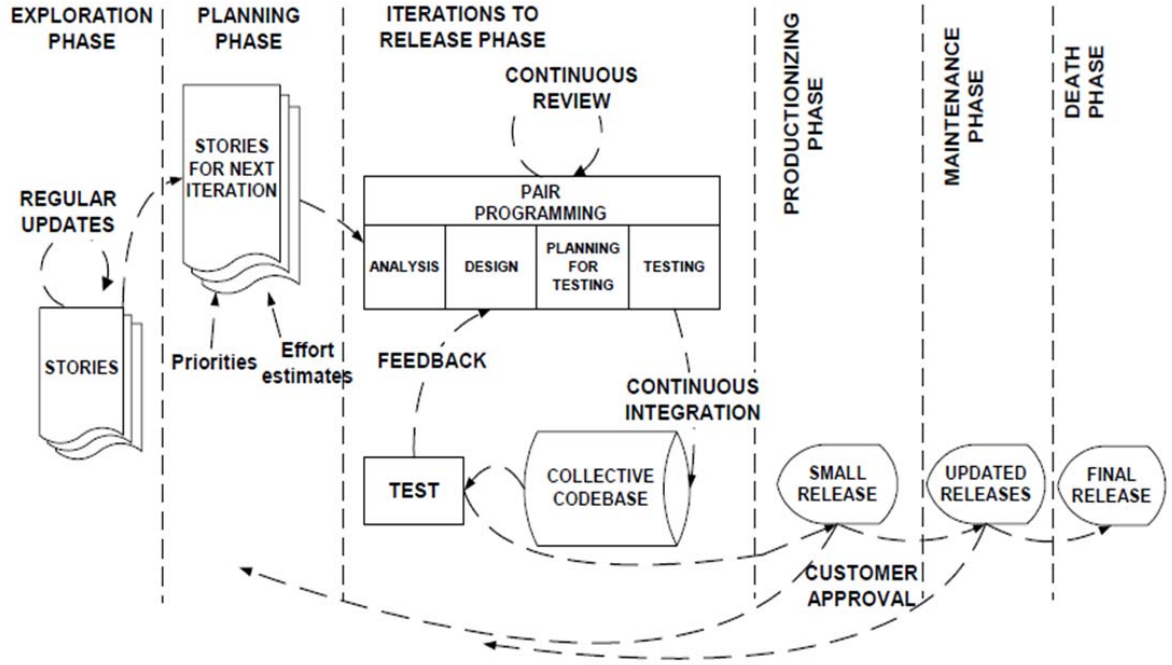

Figure 9: Extreme Programming model

2) SCRUM model

This includes a daily check in meeting to gauge process and brainstorm problems. Scrum as shown in figure 10 is a general-purpose project management framework that is applicable to any project with aggressive deadlines with complex requirements and a degree of uniqueness. [Michael J Rees, 2002].

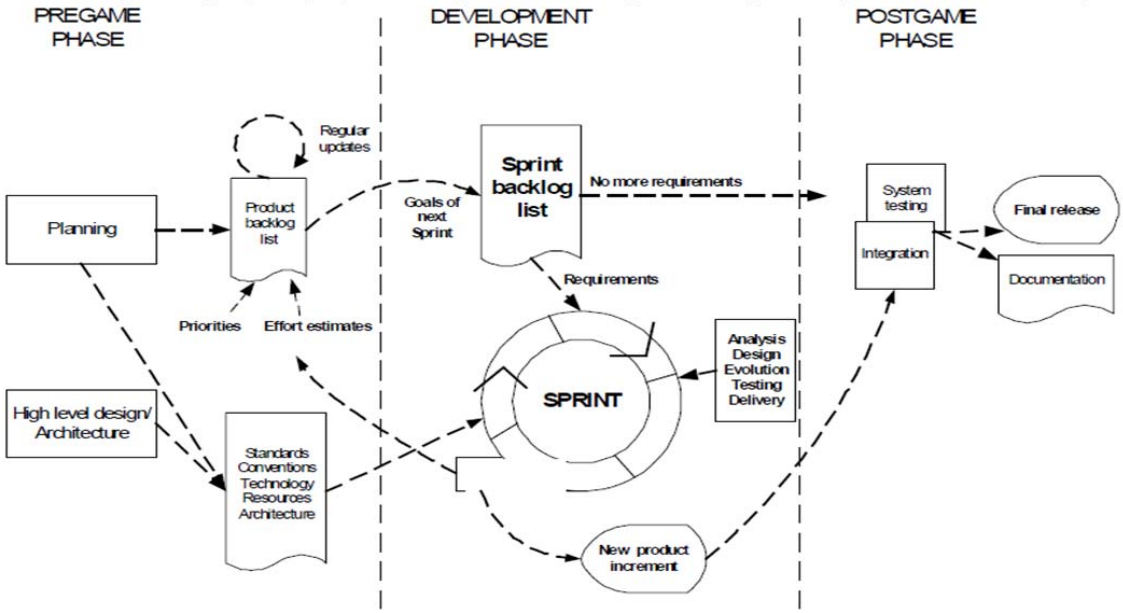

Figure 10: SCRUM model

The term Scrum originally derives from a strategy in the game of rugby where it denotes "getting an out-of play ball back into the game" with teamwork.

Scrum relies on self-commitment, self-organization, and emergence rather than authoritarian measures. [Schwaber, Ken 1996]. Scrum is suitable for small teams of less than 10 engineers. Scrum process includes 3 phases: pre-game, development and post-game.

- *Pre-game phase* includes 2 sub phases:
  - Planning includes definition of the system being developed. A product backlog list is created containing all the requirements that are currently known.
  - In the architecture phase, the high level design of the system including the architecture is planned based on the current items in the Product Backlog.
- In the *Development phase*, system is developed in Sprints which are iterative cycles where the functionality is developed or enhanced to produce new increments. Each Sprint includes: Requirements, Analysis, Design, Evolution and Delivery phases.
- The *Post-game phase* contains the closure of release including tasks such as integration, system testing and documentation.

3) Crystal Clear model

Crystal clear is an agile software development methodology developed by Alistair Cockburn. It emphasis on people, interaction, community, skills, talents, and communication as first-order effects not on tools or processes and it can be applied to teams having up to 6 developers. It can be applied to projects that are relatively small and not life critical. With Crystal clear, organizations only develop and use as much methodology as their business needs demand. Basically this is used for small teams and small projects that are not life critical. Crystal Orange is designed for medium sized projects, with a total of 10 to 40 project members and with project duration of one to two years [Gaurav Kumar et. al., 2012].

4) Feature Driven Development (FDD) model

FDD is a client-centric, architecture-centric, and pragmatic software process. There are five main activities in FDD that are performed iteratively as shown in figure 11.

- *Develop an overall model*: An overall model is developed in which the initial result will be a high-level object model and notes. At the start of a project the goal is to identify and understand the fundamentals of the domain that the system is addressing, and throughout the project this model will be refined out to reflect what is to be built.
- *Build a feature list*: grouping of features is done into related sets.
- *Plan by feature:* as the end result are a development, the identification of class owners and the identification of feature set owners is done.
- *Design by Feature*: includes detailed modeling.
- *Build by feature:* that includes programming, testing, and packaging of the system.

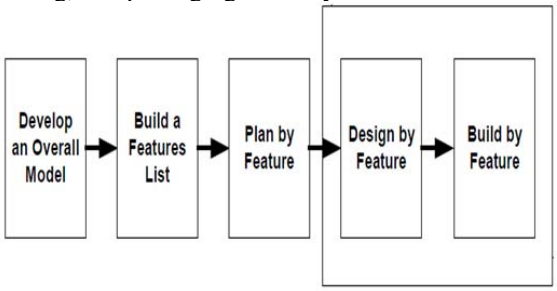

Figure 11: Feature Driven Development model

5) Test Driven Development (TDD)

Test driven development as shown in figure 12 [Scottwambler] is an approach to software development in which tests are written first then the actual development is done in iterations on the basis of those tests. Each iteration produces code necessary to pass that iteration's tests. Finally, the programmer or team refactors the code to accommodate any changes [A. Ahmed et. al., 2010].

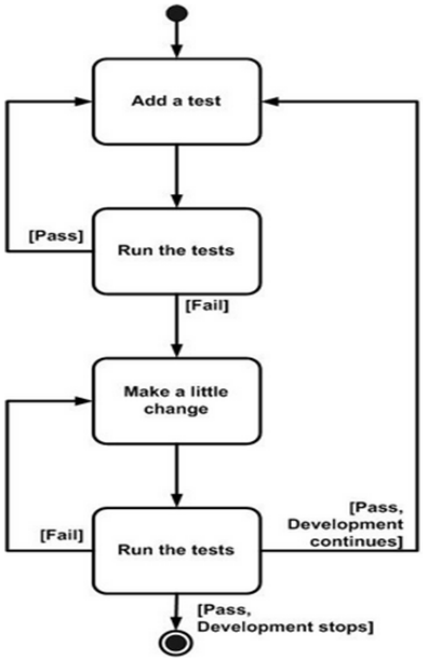

Figure 12: Test Driven Development model

Advantages of TDD are: can lead to more modularized, flexible, and extensible code; No need to invoke debugger; ability to take small steps when required. Shortcomings of TDD are: over-reliance on Unit Testing, Over testing can consume time while writing the excessive tests, and to rewrite the tests when requirements changes.

V. TRADITIONAL VS. MODERN METHODOLOGIES

Comparison of Traditional and Modern methodologies with respect to parameters is tabulated in table 1.

TABLE I. (TRADITIONAL & CLASSICAL) VS. (MODERN & POPULAR) METHODOLOGIES

Parameters/ Methodologies	**Traditional & Classical**	**Modern & Popular**
Requirements	Defined before implementation	Acquired iteratively
Rework cost	High	Low
Flexibility of Design	Hard to achieve	Easily achievable
Development direction	Fixed	Flexible
Fault detection	Problematic	Easy
Testing	After coding	On every iteration
Customer involvement	Low	High
Developers skill	Nothing in particular	Interpersonal skills & basic business knowledge
Project type	Large-scaled	Low to Medium scaled
Reusability	Hard to achieve	Easily achievable
Customer Satisfaction	Low	High
Performance	Not High	High

VI. CONCLUSION

In this paper, methodologies that examine the software development life cycle of software are studied by categorizing them into traditional and modern methodologies. Advantages and shortcomings of all of these models have been described. Traditional methods are used for highly critical projects where requirements do not change often, limited requirements with limited features, large number of developers. Modern methodologies are used for somewhat low critical projects where requirements changes often, Flexibility of design, improvement in quality, Iterative and Incremental delivery, Increased Performance, Easy fault detection capabilities. Therefore, it is important for a development team to select a software development model that best suits the project.

REFERENCES

[1] 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), Vol. 3, Issue 1, pp. 216-221, March 2013.

[2] Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham, Soo Fun Tan, "Software Development Life Cycle AGILE vs Traditional Approaches", Int. Conference on Information and Network Technology, IPCSIT Singapore, Vol. 37, pp. 162-167, 2012.

[3] Gaurav Kumar, Pradeep Kumar Bhatia, " Impact of Agile Methodology on Software Development Process", Int. Journal of Computer Technology and Electronics Engineering, Vol. 2, Issue 4, pp. 46-50, Aug. 2012.

[4] U. A. Khan, "The evolution of component based software engineering from the traditional approach to current practices", Int. Journal of Engineering and Management Research, Vol. 2, Issue 3, pp. 45-52, June 2012.

[5] Vinay Tiwari, "Software Engineering Issues in Development Models of Open Source Software", International Journal of Computer Science and Technology, Vol. 2, Issue 2, pp. 38-44, June 2011.

[6] Mostefai Mohammed Amine, Mohamed Ahmed-Nacer, "An Agile Methodology For Implementing Knowledge Management Systems : A Case Study In Component-Based Software Engineering", Int. Journal of Software Engineering and Its Applications, Vol. 5, No. 4, pp. 159-170, 2011.

[7] Nabil Mohammed Ali Munassar and A. Govardhan, "A Comparison Between Five Models Of Software Engineering", IJCSI Int. Journal of Computer Science Issues, Vol. 7, Issue 5, pp. 94-101, Sept. 2010.

[8] A. Ahmed, S. Ahmad, Dr. N. Ehsan, E. Mirza, S.Z. Sarwar, "Agile Software Development: Impact on Productivity and Quality", Proc. of IEEE ICMIT, pp. 287-291, 2010.

[9] S. Cohen, D. Dori, U. de Haan, "A Software System Development Life Cycle Model for Improved Stakeholders' Communication and Collaboration", Int. Journal of Computers, Communications & Control, Vol. 5, No. 1, pp. 20-41, 2010.

[10] Maria A. Reyes, Maryoly Ortega, Maria Perez, Anna Grimán Luis E. Mendoza and Kenyer Domínguez, "Toward A Quality Model for CBSE", Int. Conference on Enterprise Information Systems, pp. 101-106, 2009.

[11] Ying Wang, Dayong Sang, Wujie Xie, "Analysis on Agile Software Development Methods from the View of Informationalization Supply Chain Management", 3rd Int. Symposium on Intelligent Information Technology Application Workshops", pp. 219-222, 2009.

[12] "An Agile Software Development Resource", http://xprogramming.com/what-is-extreme-programming

[13] Waralak V. Siricharoen, "Ontologies and Object models in Object Oriented Software Engineering", IAENG International Journal of Computer Science, Feb. 2007

[14] "Agile Manifesto", http://agilemanifesto.org/principles.html

[15] Scottwambler, "Introduction to Test driven development", http://www.agiledata.org/essays/tdd.html

[16] K. K. Aggarwal, Yogesh Singh, Software Engineering, 3rd edition, New Age International Publishers, 2007.

[17] Nayan Jyoti Kar, "Adopting Agile Methodologies of Software Development", SETLabs Briefing, Infosys Technologies, Vol. 4, No. 1, pp. 1-9, July-Sep. 2006.

[18] Sajjad Mahmood, Richard Lai, Yong Soo Kim, Ji Hong Kim, Seok Cheon Park, Hae Suk Oh, "A survey of component based system quality assurance and assessment", Elsevier Information and Software Technology, Vol. 47, pp. 693-707, 2005.

[19] Ian Sommerville, "Software Engineering", Addison Wesley, 7th edition, 2004.

[20] Michael J Rees, "A Feasible User Story Tool for Agile Software Development", Proc. of 9th Asia-Pacific Software Engineering Conference, 2002.

[21] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, Juhani Warsta, "Agile Software Development Methods-Review and Analysis", VTT Publications 478, 2002.

[22] Daniel Robey, Richard Welke, Daniel Turk, "Traditional, iterative, and component-based development: A social analysis of software development paradigms", Journal of Information Technology and Management, Vol. 2, Issue 1, pp. 53-70, Jan. 2001.