International Journal of Mechanical Engineering and Technology (IJMET)
Volume 8, Issue 8, August 2017, pp. 967–980, Article ID: IJMET_08_08_105
Available online at http://www.iaeme.com/IJMET/issues.asp?JType=IJMET&VType=8&IType=8
ISSN Print: 0976-6340 and ISSN Online: 0976-6359

REVIEW AND COMPARISON ON SOFTWARE PROCESS MODELS

B Subbarayudu
Department of ECE, Institute of Aeronautical Engineering, Hyderabad, India

Srija Harshika D
Department of CSE, Stanley College of Engg. & Tech. Hyderabad, India

E. Amareswar
Department of ECE, MLR Institute of Technology, Hyderabad, India

R Gangadhar Reddy
Department of ECE, Institute of Aeronautical Engineering, Hyderabad, India

Kishor Kumar Reddy C
Department of CSE, Stanley College of Engg. & Tech. Hyderabad, India

ABSTRACT

Software systems come and go through a series of passages that account for their inception, initial development, productive operation, upkeep, and retirement from one generation to another. Nowadays, there are several software process models, which fulfill different purposes, approaches and requirements. However, this proliferation causes some confusion in the industry about the benefits or advantages of each proposal. In this context, studies have been conducted to determine the existing equivalence or the extent of coverage between these models having used different approaches to the comparisons. This work aims to present a study of techniques and experiences on comparison of software process models. For this study, a systematic literature review was conducted in relevant databases and available documents finding that there are few works or experiences in this area and it represents an aspect in software engineering the requires a higher level of research and development.

Key words: Software, Process Models, Deployment, Agile

Cite this Article: B Subbarayudu, Srija Harshika D, E Amareswar, R Gangadhar Reddy, Kishor Kumar Reddy C. Review and Comparison on Software Process Models, *International Journal of Mechanical Engineering and Technology*, 8(8), 2017, pp. 967–980.
http://www.iaeme.com/IJMET/issues.asp?JType=IJMET&VType=8&IType=8

1. INTRODUCTION

No one can deny the importance of computer in our life, especially during the present time. In fact, computer has become indispensible in today's life as it is used in many fields of life such as industry, medicine, commerce, education and even agriculture. It has become an important element in the industry and technology of advanced as well as developing countries. Now a day, organizations become more dependent on computer in their works as a result of computer technology. Computer is considered a time- saving device and its progress helps in executing complex, long, repeated processes in a very short time with a high speed. In addition to using computer for work, people use it for fun and entertainment. Noticeably, the number of companies that produce software programs for the purpose of facilitating works of offices, administrations, banks, etc, has increased recently which results in the difficulty of enumerating such companies. During the previous four decades, software has been developed from a tool used for analysing 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.

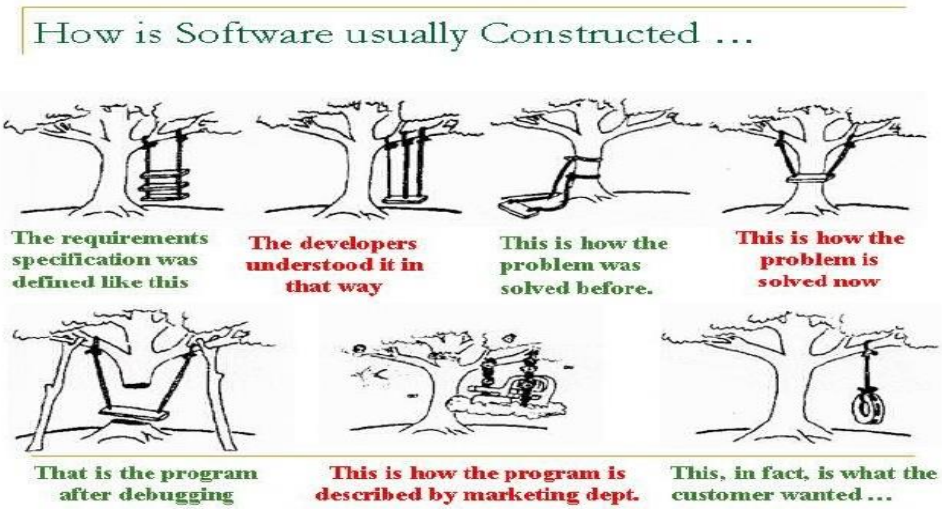

Figure 1 Construction of software

Software engineering treats the approach for developing software as a formal process much like that found in traditional engineering. Software Engineering is concerned with designing, writing, testing, implementing and maintaining software. It forms the basis of operational design and development to all computer systems. Functionality of computers is because of software. A software development process, also known as a software development life cycle (SDLC), is a structure imposed on the development of a software product. It is often considered as a subset of system development life cycle.

Computer Engineering, on the other hand, deals with studying and analyzing the algorithms and problems that are related to making the computer do the task. This involves knowing the details of how the computer as well as the network works. This field focuses more on how these computers programming languages work. Computer Engineering aims at understanding the theories that makes the computer function.

2. MYTHS OF SOFTWARE

Myth is defined as "widely held but false notation" by the oxford dictionary, so as in other fields software arena also has some myths to demystify. Pressman insists "Software myths- beliefs about software and the process used to build it- can be traced to earliest days of computing. Myths have a number of attributes that have made them insidious." So software myths prevail but though they do are not clearly visible they have the potential to harm all the parties involved in the software development process mainly the developer team.

Tom DeMarco expresses “In the absence of meaningful standards, a new industry like software comes to depend instead on folklore." The given statement points out that the software industry caught pace just some decades back so it has not matured to a formidable level and there are no strict standards in software development. There does not exist one best method of software development that ultimately equates to the ubiquitous software myths.

Primarily, there are three types of software myths, all the three are stated below:

1. Management Myth

2. Customer Myth

3. Practitioner/Developer Myth

2.1. Management Myths

Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if those beliefs will lessen the pressure (even temporarily). Some common managerial myths stated by Roger Pressman include:

- We have standards and procedures for building software, so developers have everything they need to know.
- We have state-of-the-art software development tools; after all, we buy the latest computers.
- If we're behind schedule, we can add more programmers to catch up.
- A good manger can manage any project.

The managers completely ignore that fact that they are working on something intangible but very important to the clients which invites more trouble than solution. So a software project manger must have worked well with the software development process analysing the minute deals associated with the field learning the nitty-gritty and the tips and trick of the trade. The realities are self understood as it is already stated how complex the software development process is.

2.2. Customer Myths

A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer. Commonly held myths by the clients are:

- A general statement of objectives is sufficient to begin writing programs we can fill in the details later.

- Requirement changes are easy to accommodate because software is flexible.
- I know what my problem is; therefore I know how to solve it.

This primarily is seen evidently because the clients do not have a firsthand experience in software development and they think that it's an easy process.

2.3. Practitioner/ Developer Myths

Myths that are still believed by software practitioners have been fostered by over 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard. A malpractice seen is developers are that they think they know everything and neglect the peculiarity of each problem.

- If I miss something now, I can fix it later.
- Once the program is written and running, my job is done.
- Until a program is running, there's no way of assessing its quality.
- The only deliverable for a software project is a working program.

Every developer should try to get all requirement is relevant detail to effectively design and code the system.

3. PHASES OF SOFTWARE PROCESS MODELS

Software Process model describes the phases of software cycle and the order in which those phases are executed. Each Phase produces deliverables required by next phase in life cycle.

Requirement Analysis: This is based on the requirement of project manager in the business environment. This can be analysed for their validity and possibility of incorporating the requirement in the system to be process. In this phase documentation is done for the next phase.

Design: In this phase design of software is prepared from the requirement phase. Design phase helps in defining the overall architecture of system. System design act as input for the next phase.

Implementation/Coding: After designing the work can be divided into different module and coding is started. This phase is the longest phase in process model.

Testing: After coding phase testing is done against the requirement to make sure that the product is actually solving needs. After successful testing the product is delivered / deployed to the customer for their use. Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

3.1. Brief Description of Software Process Model

Very often, process model evolution is achieved either through ad-hoc procedures or pre-defined policies. This approach is not flexible, as it offers no way of adapting evolution rules to suit individual applications. This section presents the evolution mechanism that allows different evolution strategies for the process models. Consequently, the process models should not have to submit to the same evolution constraints. The evolution semantics, during the evolution of the process models may differ according to the desired evolution strategy.

There are various Software development models or methodologies. They are as follows:

- Waterfall model
- V model
- Incremental model
- RAD model
- Iterative model
- Spiral model
- Agile model

In this section, we briefly discuss all the software development process models including the advantages and disadvantages of each model. Moreover, we also mention the different suitable cases of using these models.

3.1.1. Waterfall Model

The Waterfall Model was first Process Model to be introduced Winston Royce in 1970. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. If any error is occurred in current and previous phase then developer have to be correct it immediately that leads to less chance of error with final product. But it is highly impractical for frequently change requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In waterfall model phases do not overlap, shown in Fig.2.

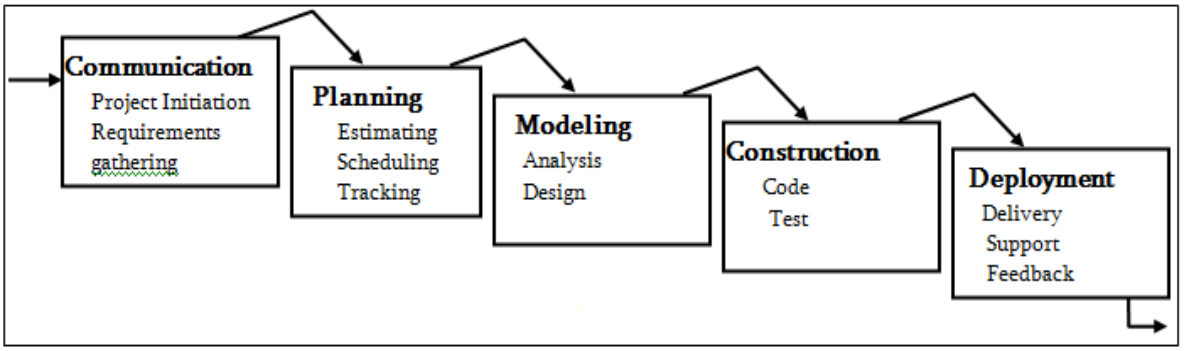

Figure 2 Waterfall Model

The steps followed in the waterfall model are:

Communication: establishes the expectations of the stakeholders and hence useful in requirements gathering.

Planning: develops a well-defined plan of execution of the project.

Modelling: develops a model of the project before developing the actual project.

Construction: builds the actual project following the plan of execution defined in the planning stage and testing.

Deployment: the delivery of end-product to the customer and its maintenance.

The advantages of waterfall model are

- Easy to understand and implement.
- Reinforces good habits: define-before-design and design-before-code.

- Identifies deliverables and milestones
- Works well on mature products and weak teams.

The disadvantages of the waterfall model are

- Real projects rarely follow the sequential approach.
- There is uncertainty at the beginning of the project regarding requirements and goals. This model does not accommodate these uncertainties very well.
- It does not yield a working version of the system until late in the process.

3.1.2. V Model

V- Model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development, shown in Fig.3.

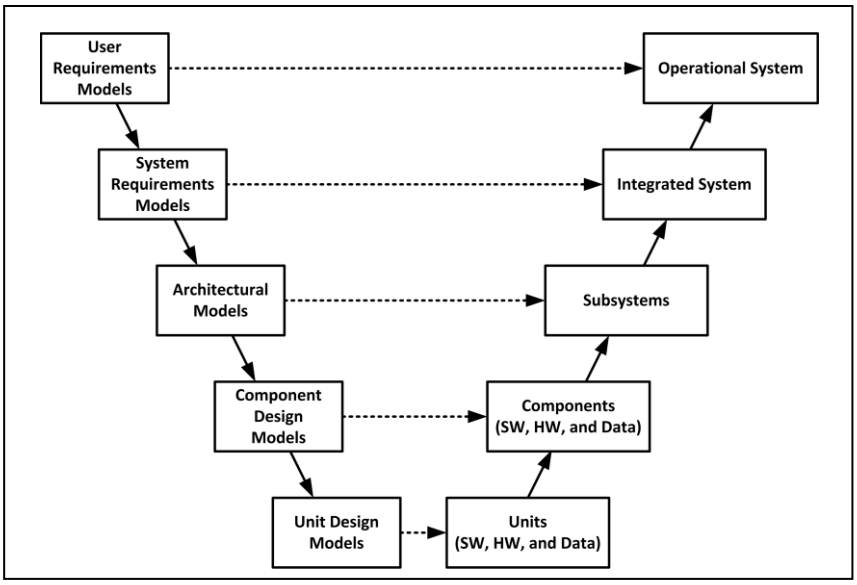

Figure 3 V Model

Advantages of V-model

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.

3.1.3. Incremental Model

The model is a method of software development where the model is designed, implemented and tested incrementally until the product is finished. The product is defined as finished when it satisfies all of its requirements. This model combines the elements of the waterfall model with the iterative philosophy of prototyping. The product is decomposed into a number of components, each of which are designed and built separately (termed as builds). Each component is delivered to the client when it is complete. This allows partial utilization of product and avoids a long development time. It also creates a large initial capital outlay with the subsequent long wait avoided. This model of development also helps ease the traumatic effect of introducing completely new system all at once, shown in fig. 3.

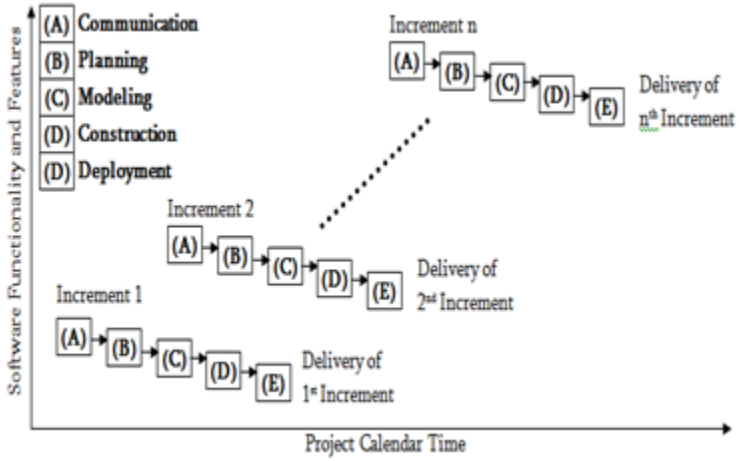

Figure 4 Incremental Model

The advantages of the incremental model are

- Divides project into smaller parts
- Creates working model early and provides valuable feedback
- Feedback from one phase provides design information for the next phase
- Very useful when more staffing is unavailable

The disadvantages of the incremental model are

- User community needs to be actively involved in the project. This demands on time of the staff and add project delay
- Communication and coordination skills take a centre stage
- Informal requests for improvement for each phase may lead to confusion
- It may lead to ―scope creep

3.1.4. RAD Model

RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements, shown in fig.4.

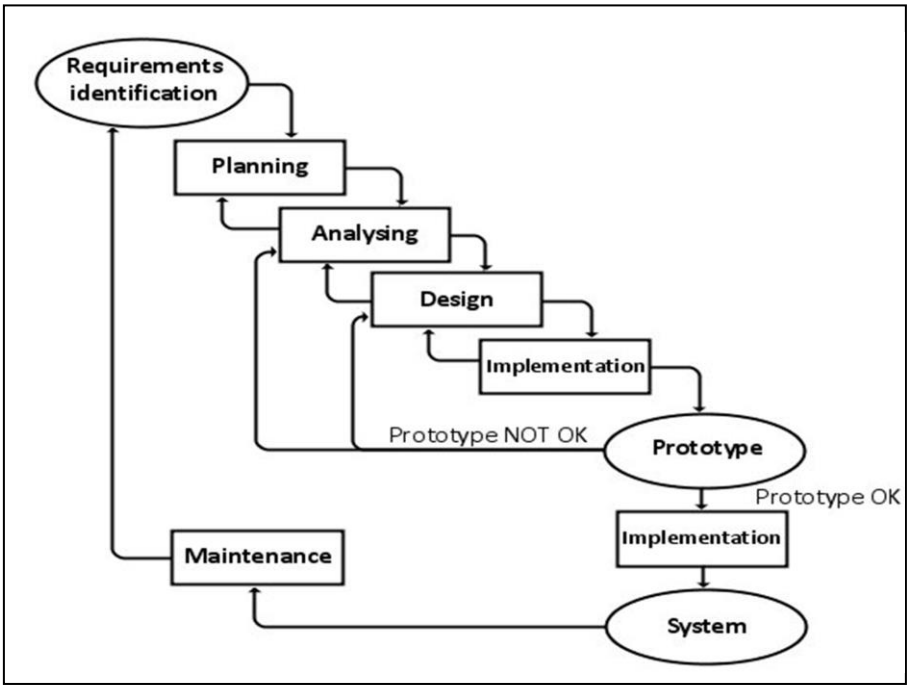

Figure 5 RAD Model

We have to use RAD model in the following cases, such as:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modelling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Advantages of the RAD Model

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

Disadvantages of RAD Model

- Depends on strong team and individual performances for identifying business requirements. Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modelling skills
- Inapplicable to cheaper projects as cost of modelling and automated code generation is very high.

3.1.5. Iterative Model

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, process begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model, shown in Fig.6.

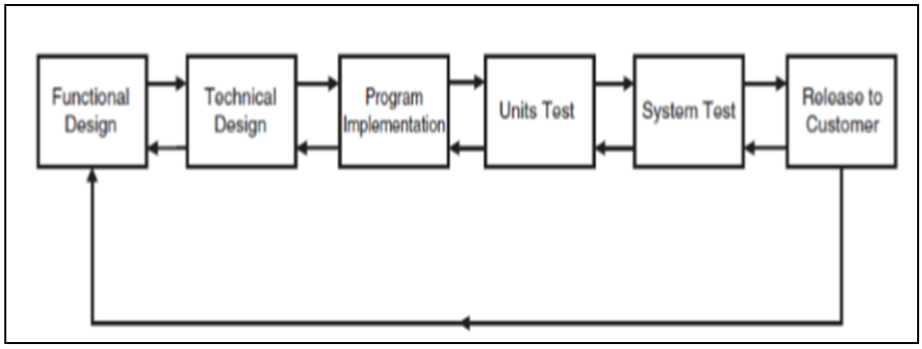

Figure 6 Iterative Model

We use Iterative model in these situations.

- Requirements of the complete system are clearly defined and understood,
- When the project is big.
- Major requirements must be defined; however, some details can evolve with time.

Advantages of Iterative Model

- In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
- In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
- In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
- In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative Model

- Each phase of iteration is rigid with no overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

3.1.6. Spiral Model

The Spiral life cycle model is similar to the Incremental model. This spiral model is developed by Boehm in 1988. It is divided into four phases: planning, risk analysis, engineering, and evaluation. A project passes through each of these phases in sequence, repeatedly, in a series of iterations called spirals. At the beginning requirements are identified for the first spiral. Spirals add functionality to this baseline spiral. Software is coded and tested during the engineering phase. During the evaluation phase, the customer has an

opportunity to evaluate the output before the project proceeds to the next spiral, shown in fig. 7.

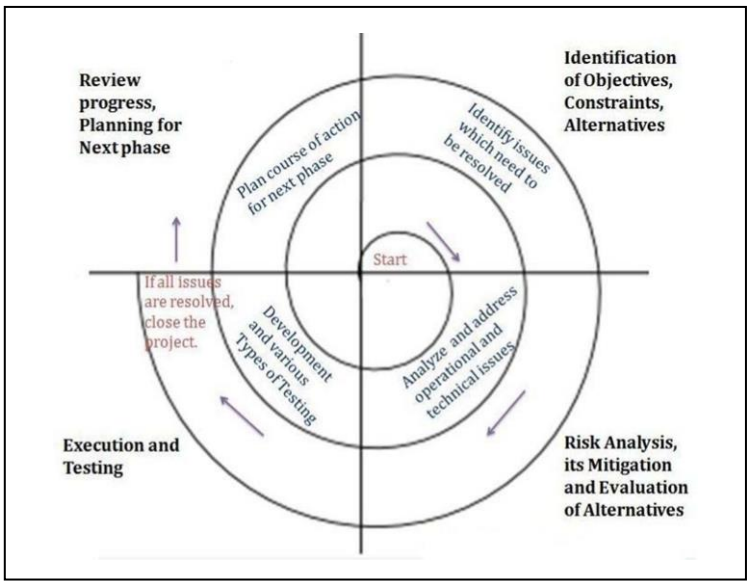

Figure 7 Spiral Model

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). In the baseline spiral, 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. A prototype is produced at the end of the risk analysis phase. Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral. In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.

A spiral model is divided into various activities which include Analysis, Design, Implementation, Testing and Deployment. The spiral is implemented in a clockwise fashion, beginning at the centre and working its way outwards, during which it passes through each of the above regions. A spiral model is divided into a number of framework activities, also called task regions.

Customer communication—tasks required to establish effective communication between developer and customer.

Planning—tasks required to define resources, timelines, and other project related information.

Risk analysis—tasks required to assess both technical and management risks.

Engineering—tasks required to build one or more representations of the application.

Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

The advantages of spiral model are

- Was designed to include the best features form Waterfall and Prototyping Model
- Good for large and mission-critical projects

- Introduces a new component – risk assessment
- Similar to prototyping model, an initial version of system is developed and modified based on input from customer

The disadvantages of the spiral model are

- Can be a costly model to use
- Risk analysis requires highly specific expertise
- Project's success is highly dependent on risk analysis phase
- Doesn't work well for smaller projects

3.1.7. Agile Model

Agile system process life cycle model has developed as a part of reaction in mid of 1990. This software will manage the problem of heavy weight, micro management of any project. Finally Agile has developed in 2001, called by lightweight method. Agile contain method like scrum, crystal cleaner. Agile use different method for different type of project but they can share common characteristic among the project. In Agile SDLC, requirement specification can change frequently because they will understand by the customer and software developer. It is found that cost of this model is very high. As the meeting of customer with software developer; this will lead to higher chance of success. Developer can control over the cost, if he acquire only needed requirement then cost can be covered by developer. This model is very difficult to implement but with low risk involvements, shown in fig.8.

Agile Process Models are:

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modelling (AM)

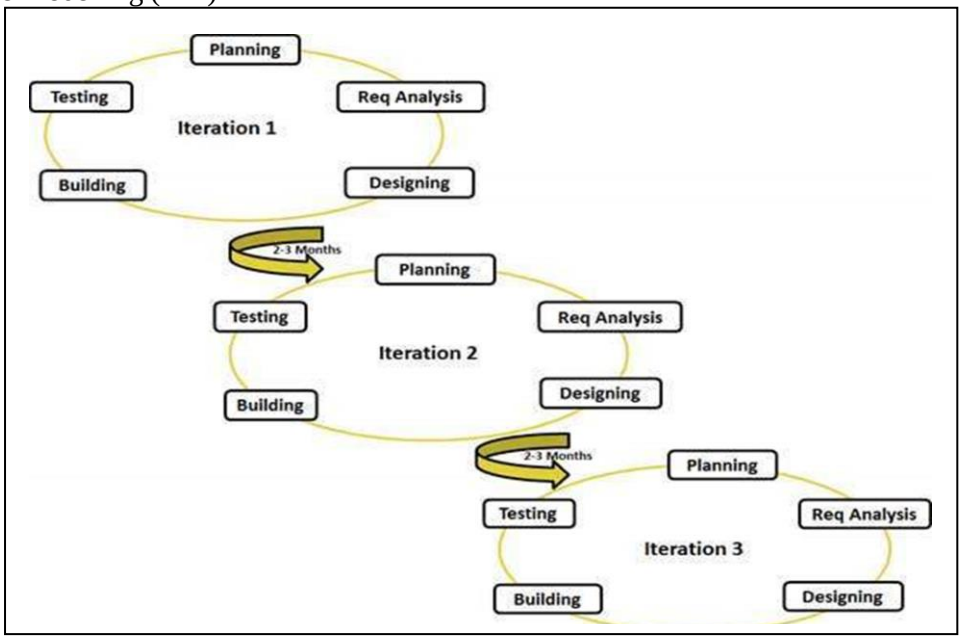

Figure 8 Agile Mode

Advantages of Agile

- Agile methodology has an adaptive team which is able to respond to the changing requirements.
- The team does not have to invest time and effort and finally find that by the time they delivered the product, the requirement of the customer has changed.
- Face to face communication and continuous inputs from customer representative leaves no space for guesswork.
- The documentation is crisp and to the point to save time.
- The end result is the high quality software in least possible time duration and satisfied customer.

Disadvantages of Agile

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

4. COMPARISION BETWEEN SOFTWARE PROCESS MODELS

Table 1

MODEL/FEATURES	WATERFALL	ITERATIVE	RAD	INCREMENTAL	SPIRAL	V MODEL
Well Defined Requirements	Yes	No	Yes	No	No	Yes
User Involvement in all phases	Only at beginning	High	Only at beginning	Yes (Intermediate)	High	No
Risk Analysis	Only at beginning	No Risk Analysis	Low	No Risk Analysis	Yes	Only at beginning
Overlapping Phases	No Overlapping	Yes	No	No	Yes	No
Implementation Time	Long	Quick	Quick	Long	Long	Long
Cost	Low	High	Low	Low	Expensive	Expensive
Incorporation of changes	Difficult	Easy	Easy	Easy	Easy	Difficult
Simplicity	Simple	Simple	Simple	Intermediate	Intermediate	Intermediate
Flexibility	Rigid	Little Flexible	High	Less Flexible	Flexible	Less Flexible

5. CONCLUSIONS

Nowadays, there are several software process models, which fulfil different purposes, approaches and requirements. However, this proliferation causes some confusion in the industry about the benefits or advantages of each proposal. In this context, studies have been conducted to determine the existing equivalence or the extent of coverage between these models having used different approaches to the comparisons. After completing this research, it is concluded that:

- There are many existing models for developing systems for different sizes of projects and requirements.
- These models were established between 1970 and 1999.
- Waterfall model and spiral model are used commonly in developing systems.
- Each model has advantages and disadvantages for the development of systems , so each model tries to eliminate the disadvantages of the previous model

REFERENCES

[1] A. Finkelstein, J. Kramer and B. Nuseibeh (Eds.), Software Process Modelling and Technology, pp. 187-222, Research Studies Press, Taunton, England, 1994.

[2] R. Conradi and M.J. Jaccheri. “Process Modelling Languages”. In J.C. Derniame, B.A. Kaba and D. Wastell (Eds.), Software Process: Principles, Methodology and Technology, pp.27-51, No. 1500, LCNS, Springer-Verlag, 1999.

[3] L. Osterweil. “Software Processes Are Software Too”. In Proc. of the 9th Intl. Conf. on Software Engineering, pp. 2-13, Monterey, CA, March 1987. IEEE Computer Press

[4] J.M. Ribo and X. Franch. “PROMENADE: A PML Intended to Enhance Standarization, Expressiveness and Modularity in Software Process Modelling”. Research Report LSI-00-34-R, Llenguatges I Sistemes Informatics, Politechnical of Catalonia, 2000.

[5] I. Podnar, B. Mikac and A. Caric. “SDL Based Approach to Software Process Modeling”. In R. Conradi (Ed.), Proc. of 7th European Workshop on Software Process Technology (EWSPT 2000), pp. 190-202, Kaprun, Austria, February 2000. Springer

[6] B. Balzer, “Tolerating inconsistencies,” presented at International Conference on Software Engineering (ICSE 13), Austin (TX), 1991

[7] V. Ambriola, R. Conradi, and A. Fuggetta, “Assessing process-centered software engineering environments,” ACM Transactions on Software Engineering and Methodology, vol. 6, 1997.

[8] W. S. Humphrey, A discipline for Software Engineering: Addison-Wesley Publishing Company, 1995.

[9] A. S. Lee, “A scientific methodology for MIS case studies,” MIS Quartely, vol. 13, pp. 33-50, 1989.

[10] Sanjana Taya, Shaveta Gupta, “Comparative Analysis of Software Development Life Cycle Models”.

[11] 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

[12] Roger Pressman, titled “Software Engineering - a practitioner's approach”

[13] Klopper, R., Gruner, S., & Kourie, D. (2007), “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.

[14] Dr. S. Ravichandran, Literature Review On Software Metrics For Software Process Improvement (SPI), International Journal of Information Technology & Management Information System (IJITMIS), 2(1), 2011, pp. 01-04

[15] Swarnalatha K S, G.N Srinivasan, Pooja S Bhandary, A Constructive And Dynamic Frame Work For Requirement Engineering Process Model – BEE HIVE Model, International Journal of Computer Engineering and Technology, 5(7), 2014, pp. 48-54.