



Comparison of Software Life Cycle Models

¹Prateek Thakral, ²Saurabh Sharma

Department of Computer Applications
NIT, Kurukshetra, Haryana, India

Abstract: Computer software continues to be the single most important technology on the world stage. Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines. Software development organizations follow some process when developing a software product. A variety of life cycle models has been proposed and is based on tasks involved in developing and maintaining software. This paper presents a comparison between various software life cycle models used for software development along with their advantages and disadvantages.

Keywords: Software, Software Engineering, Waterfall, Prototype, Spiral, SDLC

I. INTRODUCTION

The nature and complexity of software have changed significantly in last few decades. Software has become critical to advancement in almost all areas of human endeavor. The computer industry has also progressed at a break-neck speed through the computer revolution. Today, software takes on a dual role. It is a product, and at the same time, the vehicle for delivering a product. Whether it resides within a mobile phone or operates inside a mainframe computer, software is information transformer- producing, managing, acquiring, modifying, displaying or transmitting information that can be as simple as a single bit or as complex as multimedia presentation. Software can be defined as instructions or computer programs that when executed provide desired features, functions and performance or data structures that enable the programs to adequately manipulate information [1]. Software is developed or engineered, it is not manufactured. Software doesn't wear out. Software engineering is simply a discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements. A generic process framework for software engineering consists of five activities: Communication, Planning, Modeling, Construction and Deployment [2, 3]. These activities can be used during software development using any software life cycle model. The details of software process may be different, but framework activities remain the same. This paper gives a detailed view of various life cycle models used for software development (SDLC). The rest of the paper is organized as follows. Section II discusses the various life cycle models used for software development in detail. Section III presents comparison between the various models depending upon various parameters. Finally, conclusion is done in section IV.

II. SOFTWARE LIFE CYCLE MODELS

The existence of software process is no guarantee that software will be delivered on time, meet customer's need, or it will exhibit the technical characteristics. Process patterns must be coupled with solid software engineering practices. A software life cycle is the series of special stages that a software product undergoes during its life time. A software life cycle model (SDLC) is a descriptive and diagrammatic representation of the software life cycle [4]. It represents all the activities required to make a software product travel through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its commencement to retirement. The primary advantage of adhering to a life cycle model is that it encourages development of software in a systematic and disciplined manner. Without software life cycle models, it becomes difficult for software project managers to monitor the progress of the project. A variety of life cycle models has been proposed and is based on tasks involved in developing and maintaining software.

The most familiar model is the waterfall model [4, 5]. The name of this model is justified by its diagrammatic representation which resembles a cascade of waterfall. This model divides the software life cycle in phases as shown below.

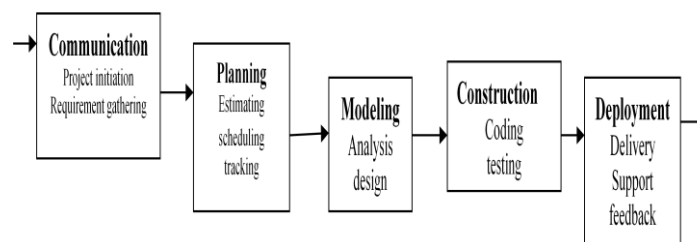


Fig. 1 Waterfall model

In this model, a project begins with feasibility analysis. Upon successfully demonstrating the feasibility of a project, the requirements analysis and project planning begin. The design starts after the requirements analysis is complete, and coding begins after the design is complete. Once the programming is completed, the code is integrated and testing is done. Upon successful completion of testing, the system is installed. After this, the regular operation and maintenance of the system takes place. The basic idea behind the phases is separation of concerns—each phase deals with a distinct and separate set of concerns. By doing this, the large and complex task of building the software is broken into smaller tasks of specifying requirements, doing design, etc. Separating the concerns and focusing on a select few in a phase gives a better handle to the engineers and managers in dealing with the complexity of the problem.

Next model is prototyping model [6]. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determine the requirements. A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype usually turns out to be a very crude version of the actual system.

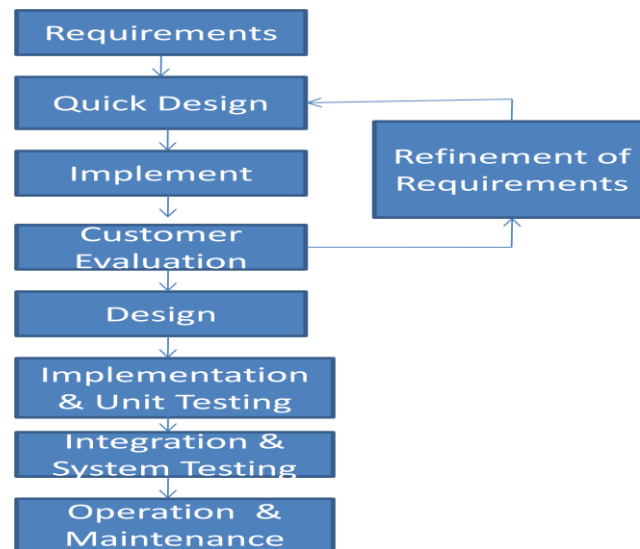


Fig. 2 Prototyping Model

The development of the prototype typically starts when the preliminary version of the requirements specification document has been developed. At this stage, there is a reasonable understanding of the system and its needs and which needs are unclear or likely to change. After the prototype has been developed, the end users and clients are given an opportunity to use and explore the prototype. Based on their experience, they provide feedback to the developers regarding the prototype: what is correct, what needs to be modified, what is missing, what is not needed, etc. Based on the feedback, the prototype is modified to incorporate some of the suggested changes that can be done easily, and then the users and the clients are again allowed to use the system. This cycle repeats until, in the judgment of the prototype developers and analysts, the benefit from further changing the system and obtaining feedback is outweighed by the cost and time involved in making the changes and obtaining the feedback. Next comes the Iterative development model. The basic idea is that the software should be developed in increments, each increment adding some functional capability to the system until the full system is implemented [8]. The common approach for iterative development is to do the requirements and the architecture design in a standard waterfall or prototyping approach, but deliver the software iteratively.

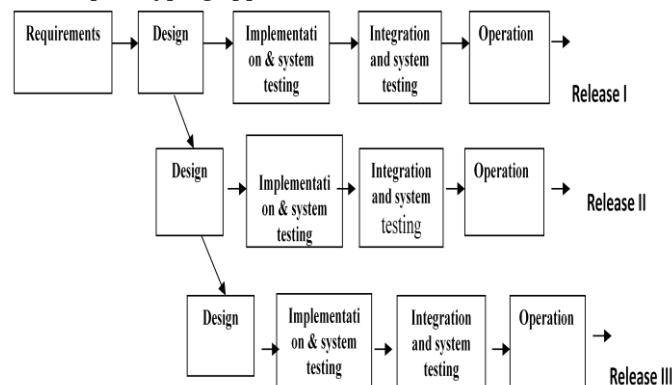


Fig. 3 Iterative Enhancement model

That is, the building of the system, which is the most time and effort-consuming task, is done iteratively, though most of the requirements. The iterative approach is becoming extremely popular, despite some difficulties in using it in this context. The advantage of this approach is that as the requirements are mostly known upfront, an overall view of the system is available and a proper architecture can be designed which can remain relatively stable.

Originally proposed by Barry Boehm, the spiral model [9] is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. Each phase in this model is split into four sectors (or quadrants) as shown in fig 4. The following activities are carried out during each phase of a spiral model.

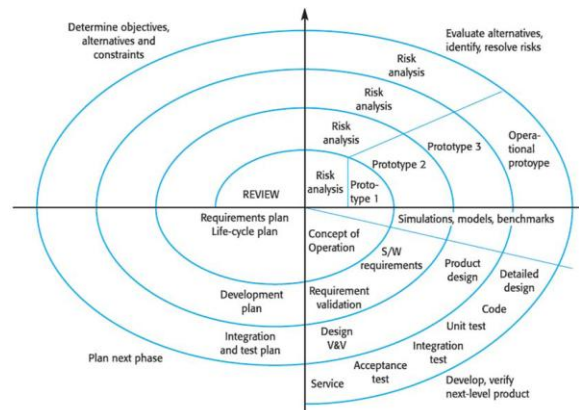


Fig. 4 Spiral Model

During the first phase, planning is performed, risks are analyzed, prototypes are built, and customers evaluate the prototype. During the second phase, a more refined prototype is built, requirements are documented and validated, and customers are involved in assessing new prototype. By the third phase, risks are known, and a more traditional development approach is taken. The focus is the identification of the problems and the classification of these into different levels of risks. The spiral model is a realistic approach to the development of large-scale systems and software. The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model.

III. COMPARISON OF LIFE CYCLE MODELS

The waterfall model is intuitively the most obvious way to develop software. The waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. The waterfall model [5, 9] is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. The classical waterfall model cannot be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases. This problem is overcome in the iterative waterfall model [10]. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However, this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks.

The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects. The reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product. A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

The aim of waterfall model and prototyping model is the delivery of a complete, operational and good quality product. In contrast, the iterative enhancement model [11] does deliver an operational quality product at each release, but one that satisfies only a subset of customer's requirement. The complete product is divided into releases, and the developer delivers the product release by release. A typical product will usually have many releases as shown in fig 3 above. With this model, first release may be available within few months, whereas the customer generally waits months or years to receive a product using the waterfall model and prototyping model. The spiral model is called a Meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model [12]. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. The advantage of this model is the wide range of options to accommodate the good features of other life cycle models. The risk analysis and validation steps eliminate errors in the early phase of development. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects

IV. CONCLUSION

In this paper, a comparative study of various software life cycle models has been done. The waterfall model, prototyping model, iterative enhancement model and spiral model are compared along with their advantages and disadvantages. Each model has its own importance and quality under different conditions and situations. Waterfall model follows a linear approach for software development which is easy to implement. The waterfall model is the basic model and all other life cycle models as embellishments of this model. The prototyping and spiral model comes under evolutionary process model category which is best suitable for making large products. The spiral model combines the linear approach of waterfall model and iterative approach of prototype model & solves the risk factor problem which wasn't included in any other life cycle model. The iterative enhancement model is well suited for developing software into releases time to time which satisfies a particular subset of customers as according. This model develops and delivers software product in much less time as compared to various other life cycle models.

REFERENCES

- [1] Pressman, R. S., 1998, Software Project Management. In: Software Engineering – A Practitioner's Approach, Fourth Edition, pp. 28-38. McGraw- Hill Companies Publication, Inc. USA.
- [2] Scacchi, W., 2001, *Process Model in Software Engineering*, J. J. Marciniak ed., *Encyclopedia of Software Engineering*, 2nd Edition, John Wiley and Sons, Inc, New York, December 2001.
- [3] W.S. Humphrey, Characterizing the Software Process: A Maturity Framework, Software Engineering Institute, CMU/SEI-87-TR-11, ADA182895, June 1987.
- [4] 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.
- [5] Mall, R., 2004 Classical Waterfall Model". In: "Fundamentals of Software Engineering", p.p. 19-26.
- [6] K. K. Aggarwal, Yogesh Singh, Software Engineering, 3rd edition, New Age International Publishers, 2007.
- [7] ISO/ IEC 12207:1995, Information technology –Software life cycle processes, International Standards Organization.
- [8] Craig Larman and Victor Basili, "*Iterative and Incremental Development: A Brief History*", IEEE Computer, 2003
- [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] Nayan Jyoti Kar, "Adopting Agile Methodologies of Software Development", SETLabs Briefing, Infosys Technologies, Vol. 4, No. 1, pp. 1-9, July-Sep. 2006.
- [11] Alfonso Fuggetta, "Software Process: A Roadmap", Proc. 22nd Int'l Conf. Software Eng.-Future of Software Eng., 2000, pp. 25-34.
- [12] ISO/IEC 12207:1995, Information technology –Software life cycle processes, International Standards Organization.