Feasibility Study on Software Engineering Models

Pratik Sankar Panda, Sunit Gourav Mohanty, Monisa Nayak, Romit Panigrahi';

;

Abstract— **Software Engineering is an application of engineering principles in software developments. The principles can be defined as by applying well proven techniques in an organized and systematic manner. The objective is to develop a high quality software product. The software contains a set of codes, a set of manuals which includes user manual and design manual. User manual is the one in which what kind of services are provided to the users of the system depending on the user type. Software engineering helps to produce or develop high quality software within the available budget and time. There are a few attributes defining the software quality like usability, maintainability, reliability, portability, reusability and efficiency**

Index Terms— **Software Engineering, Software Systems, Software Design, Designing Models**

I. INTRODUCTION

In modern days we have to face many complex situations in the latest software systems. It would be very difficult to overcome these problems without any proper path or strategy being followed. By putting use of management and evolution of intensive system service the advancement of software engineering community has taken place. In the operational context, these complexities have lead to resilient energy efficiency which depends on customizable and adaptable changes. It consists of four P's such as Product, Process, People, and Project. In order to get the product we do various activities and those activities are referred as a process. Software is incorporeal because it is not composed of matter; having no material existence. It is an unsubstantial way of approach. When a human-readable programming language (preferably text) describes computers instructions in a bulk it is called source code. There are various models implemented for designing of process model like Waterfall model (Iterative Waterfall model), prototype model, Spiral Model (Verification and Validation), Evolutionary Model and RAD Model. These are various targets or goals of software engineering which includes the improvement and upliftment of the standard and productivity of the outcome. It also takes care of the job satisfaction of the software engineer. As per the distribution effort of software the life span is 1-3 years in developing it and 5-15 years for its maintenance. Distribution between development and maintenance are likely to be 40/60, 30/70 to 10/30. Maintenance includes Corrective, Adaptive, and Perfective which states that it is very likely that defects may be seen by the customers and those defects can be corrected as per the maintenance of the software. This result in modification of software's to accommodate changes in its surroundings. As the customer recognizes the additional functions in a software that will provide benefits to the individuals to extend the software's functionality requirement.

Manuscript received April 17, 2017

Pratik Sankar Panda, Department of Computer Science and Engineering, Siksha 'O' Anusandhan University, Bhubaneswar, Orissa
Sunit Gourav Mohanty, Department of Mechanical Engineering, Siksha 'O' Anusandhan University, Bhubaneswar, Orissa
Monisa Nayak, Department of Computer Science and Engineering, Siksha 'O' Anusandhan University, Bhubaneswar, Orissa
Romit Panigrahi, Depatment of Electrical Engineering, Siksha 'O' Anusandhan University, Bhubaneswar, Orissa

II. SOFTWARE DESIGN MODELS

2.1 SDLC

Basically present days for design, development and testing high quality software IT sectors uses a framework with predefined task management called development life cycle (SDLC). These are being done so that we can determine a proper model to be used so that we get a good quality product with minimal time and cost . it helps in a systematic approach to a problem . There are various software development models each having its specific way of designing and execution of the problems. These models are to be selected on the basis of the software problem being generated and the best solution that can be provided to it.

2.2 Classical (Iterative Waterfall Model)

Basically it describes a simple model to implement and manage. Each of its phases has entry and exit criteria. After the completion of one phase the next phase starts there is no overlapping of phases. A review is conducted during each phase in order to check that the project is going on in a right direction. Some deliverables are also prepared in each of its phases. The model is very rigid.

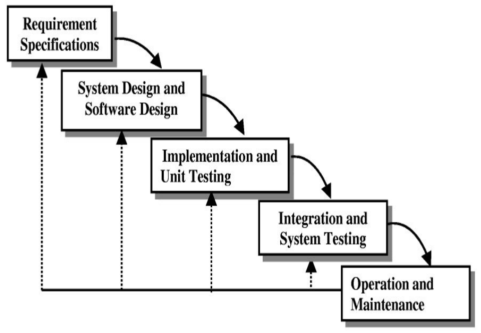
Requirement Specifications
System Design and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Figure 1: Classical (Iterative Waterfall Model)

In Iterative waterfall model the backward paths from every phase to its previous phase is considered as in addition to the classical waterfall model. This path allows for correction of errors committed during a phase as and when they are detected in a later phase. However, there are no feedback paths for feasibility study and hence feasibility study errors cannot be detected. The principle by which errors are detected as close to introduction point as possible is called phase containment of errors. We often see phase containment of errors to achieve reviews of being conducted after every milestone. In this model, the phases overlap in time as a phase may not end at a specific time instant if we include the rework required for the phase later due to errors in the phase detected in the later stage.

2.3 Prototype Model

Whenever we are required to build up a design of a software system a working prototype is needed. The toy implementation of a system (a prototype) with a few functionalities so as to illustrate the customer the formats and layout of the design messages etc. Many technical issues are associated with the developing process and also many design decisions are dependent on the efficiency of a sorting algorithm.

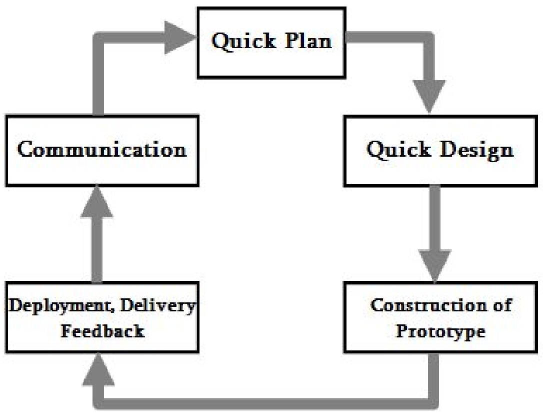

Figure 2: Prototype Model

This model is more user friendly because it gives the user a chance to interact with the dummy system before the actual system is built up. The missing functionalities can also be identified easily and the developer can design or reframe the structure as per requirement. It has more chances of giving an accurate output at the end.

2.4 Rapid Application Development Model (RAD)

The rapid prototyping can be minimized by a software development methodology. For faster product delivery basically RAD model functional modules are developed in parallel to make the complete product. It follows an iterative as well as incremental model consisting of developers, domain experts, customer representatives, IT management working gradually on their prototype.

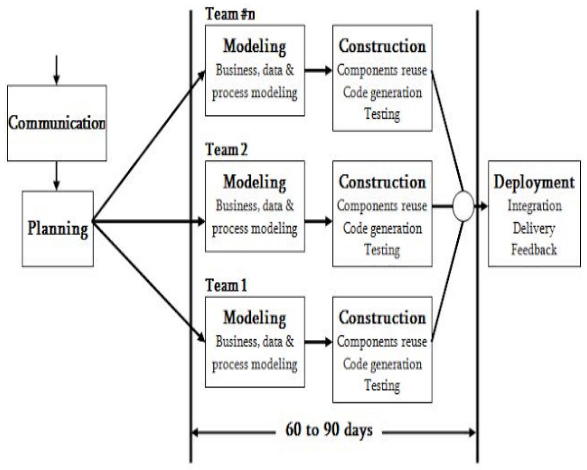

Figure 3: Rapid Application Development Model (RAD)

Business modeling, Data modeling, Process modeling, Testing and Turnover are the various phases of RAD Model. Business modelings are designed in terms of the information flow and information distribution among various business channels. By driving successful flow of information a complete business analysis is performed. It helps to investigate the vital business information, factors can be obtained. The data modeling is the information collected in business modeling phase for reviewed and analyzed data objects form sets vital for the business. The data attributes sets are identified and defined relation between data objects.
In the case of process modeling the data objects sets establish the business information flow (defined in the modeling phase) are converted to gain the specific business objectives as per the business model. Process descriptions are required for summing, removing, retrieve or modify data objects in this phase. By using automation tools and data models into actual prototypes in an application generation helps to build the actual system and coding layout. The overall testing time is reduced in the RAD model by testing and turnover of prototypes independently. Interfaces testing between prototypes should be done properly. The overall risk is reduced as programming components have been already tested thoroughly.

2.5 Evolutionary Model

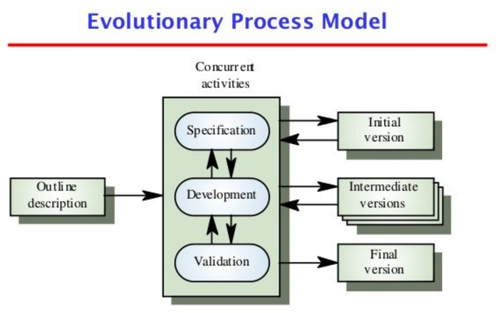

Figure 4: Evolutionary Process Model

In this evolutionary model is very natural model to use in object oriented software development projects. The user gets a chance to model first a simple working system is built which subsequently undergoes many functional improvements and additions until a desired system is built. This model is known as "Design a little, build a little, test a little, and deploy a little model". After the requirements have been specified in the above figure 4 are software modules of the product which are incrementally developed and delivered. Software requirements are first broken down into several modules and the core module is first developed. This experiment done with moderately developed software much before software's complete version is released. This model is helpful to know the requirements of requested by the customer after the software is delivered becomes very less.

2.6 Verification and Validation Model

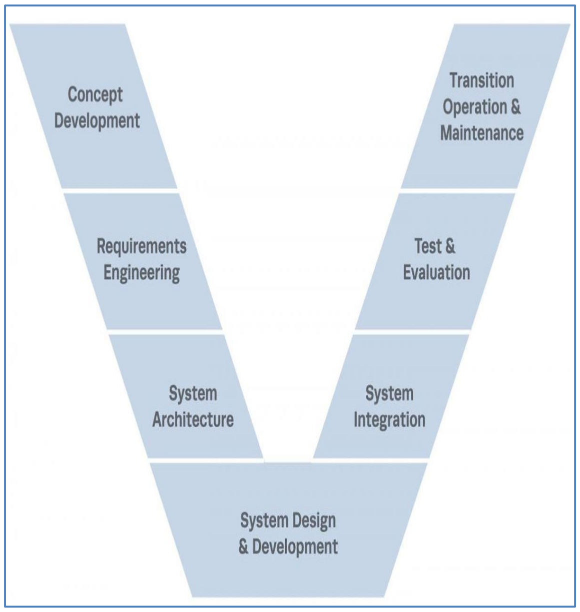

Figure 5: Verification and Validation Model

This model deals with processes execution taking place in a sequential manner in V-shape format. It is the advanced model of waterfall model where for each development phase has its own testing phase as of which whenever a level testing phase has been successfully completed then only we can go for the next development phase the corresponding testing phase of the development phase and parallel model is used for the verification phases on one side along with other side validation phases. Both the sides of the V-model are connected by coding phase. A well defined and clearly documented requirement is being acknowledged for this V-model. There is a stable product definition assigned. The technology has ambiguous requirements as well it is not dynamic but it is well understood by the project group.

2.7 Spiral Model

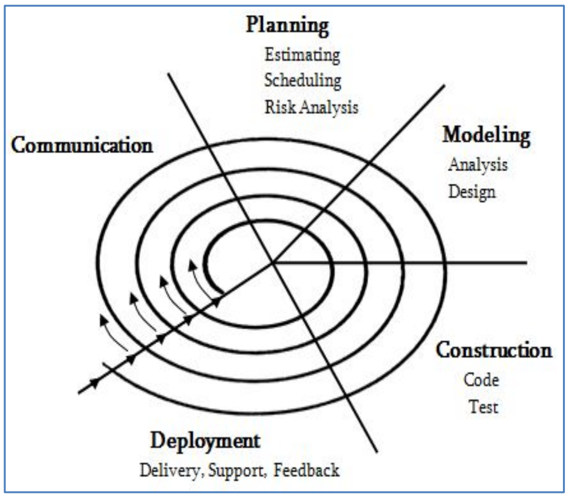

Figure 6: Spiral Model

In this model each phase represents a phase of the software process like requirement analysis, system design. The inner most loops are linked with system feasibility where as the next loop of the system lies on requirements of the system and the next one is linked with the design of the system. In this model we have no fixed phases. According to the figure, the first quadrant is for determining objectives. The phase objectives are identified here and objectives risk are also examined here. Risk is any adverse circumstance which slows or hinders the successful software project completion. The second quadrant aims at resolving risk identified. The analyst task is to verify and eliminate the single identified project risk commonly called the practice of risk assessment and reduction. The third quadrant represents the development and validation of the next level of the product with risk reduced. The fourth quadrant is needed for review and planning. The customer reviews the results achieved before the next iteration planning around the spiral is done. Spiral model acts as an "Meta model" as it matches with other models in one way or other. The complete version of software gets progressively built with each iteration the spiral. Here waterfall model is represented by a single loop of spiral. It uses iterations (an evolutionary approach) through the spiral are evolutionary levels. It enhances the understanding and risk response during every spiral iterations which is equal to iterative waterfall model. Prototyping has a risk reduction mechanism that is similar to the aspects of prototype model.

III. Conclusion

As we have already came to know about the various software design models and their applications in the developing of software it would be beneficial for us in the better understanding and implementation of the appropriate model as and when required. These models help us to think design and implement the model in such a way that we get our outcome in a simplified manner. Whenever we go for these models we get more accurate results and the defects reduce in the final code. It increases the modularization and decomposition of the system. It also enables the use of parts of the system in the new project. We should never forget that using of any wrong model to do a task could not only damage the entire system but also lead to wastage of resources so it should be taken high care that selection of the model should be correct. Previously which was very difficult to achieve could be done by these models.

References

[1]Pohl, K., Böckle, G. and van Der Linden, F.J., 2005. Software product line engineering: foundations, principles and techniques. Springer Science & Business Media.

[2]Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M. and Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. Journal of systems and software, 80(4), pp.571-583.

[3]Lyu, M.R., 1996., Handbook of software reliability engineering.

[4]Herbsleb, J.D. and Moitra, D., 2001. Global software development. IEEE software, 18(2), pp.16-20.

[5]Musa, J.D., 1975. A theory of software reliability and its application. IEEE transactions on software engineering, (3), pp.312-327.

[6]Glass, R.L., Vessey, I. and Ramesh, V., 2002. Research in software engineering: an analysis of the literature, Information and Software technology, 44(8), pp.491-506.

[7]Huang, C.Y., 2005. Performance analysis of software reliability growth models with testing-effort and change-point. Journal of Systems and Software, 76(2), pp.181-194.

[8]Sametinger, J., 1997. Software engineering with reusable components. Springer Science & BusinessMedia.

[9]David, O., Ascough, J.C., Lloyd, W., Green, T.R., Rojas, K.W., Leavesley, G.H. and Ahuja, L.R., 2013. A software engineering perspective on environmental modeling framework design: The Object Modeling System. Environmental Modelling & Software, 39, pp.201-213.

[10] Walton, G.H. and Poore, J.H., 2000. Generating transition probabilities to support model-based software testing. Software: practice and experience, 30(10), pp.1095-1106.

[11] Panichella, A., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D. and De Lucia, A., 2013, May. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In Proceedings of the 2013 International Conference on Software Engineering (pp. 522-531). IEEE Press.

[12] Miller, J., 2000. Applying meta-analytical procedures to software engineering experiments. Journal of Systems and Software, 54(1), pp.29-39