DOI: http://dx.doi.org/10.26483/ijarcs.v8i8.4839

ISSN No. 0976-5697

Volume 8, No. 8,September-October 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

SIGNIFICANCE OF SOFTWARE DEVELOPMENT MODELS

Rayan Dasoriya
Department of Computer Engineering
SVKM's NMIMS Mukesh Patel School of Technology Management and Engineering
Mumbai, India

Abstract: With the evolution of different software development models over the past years, it became a topic of utmost interest to categorize and segregate them depending upon the applications, advantages and disadvantages. There are various factors that affect the projects. They should also be taken care of when we select a development model. Many software projects fail due to reasons like an unskilled developer, time limit, poor quality, less user involvement and much more. Software development models should be selected wisely looking at the conditions and quality of the developer, user, time and complexity of the project. All these factors play a vital role in the success of the project. Models can be categorized as light weight models and heavy weight models. This paper discusses various models on different metrics with pros and cons of each of them and also help us select an appropriate model depending upon the project.

Keywords:Software Development; SDLC; Waterfall model; Agile Model; Incremental model; Prototype model

I. INTRODUCTION

Software is a term whose definition has evolved over the time. With the passing years, the definition of software was changing. Some of the proposed changes were accepted and some were rejected. After that, a definition was suggested which gives us detail about software. It describes software as a set of different functionalities [1]. These include:

- Set of programs/instructions that when executed provide desired features, function and performance
- Data structure that enables the program to manipulate the information adequately.
- Information in the form of hard copy and virtual forms that describe the use and operations of programs.

Unlike a physical system element, software is a logical element. Further elements can be discovered to update the definition of the software. It differs from the hardware in the following manner:

- Rather than being manufactured, it is developed or engineered.
- Software doesn't deteriorate with time.
- Most of the software continue to be custom-built rather than component based construction.

Software Development, on the other hand, comprises of programming, documenting, testing and bug fixing in the creation and maintenance of applications which in turn result in a software product. It is a planned and structured process of the development of the desired software from the conception to the final manifestation.

The role of software applications resembles in different fields which include banking, communication, marketing, transportation, education, etc. The software product is usually developed in a series called life cycle.

SDLC or Software Development Life Cycle [2] consists of a series of steps for developing and designing a software product efficiently. Different models provide different life cycles for a project to be completed and this can be determined depending upon various factors which include team size, cost, time, risk management etc.

In the earlier days, a code was written and then debug. But this approach was unsuccessful when it was implemented to large projects. This led to the discovery of waterfall model. This model constitutes of various stages from communication, planning, modeling, construction and deployment. It is also known as a classic life cycle. It suggests a sequential and systematic approach for developing a software according to the need of specification provided by the customer. This model is useful when well define enhancements to the existing software is to be made. It is used only when the requirements are well defined and stable.

Another model is the spiral model [3] in which a software is developed in a series of releases. In the earlier stage, the versions of the product may be used as a prototype. Later on, complete and more complex versions are released. In the spiral model, as the iterations increase, we come near our target and with subsequent passes in the spiral, we are progressively improving our system. Risk is also considered as each evolution is made. All the milestones that are achieved during a spiral are noted down and it is referred as anchor point milestones. This model can be applied to the software even after it is delivered.

The incremental model applies linear sequences as the calendar time progresses. Each deliverable sequences increments in a similar manner as the evolutionary process flow produces. The first increment is usually produces a core product. All the basic requirements are mentioned but many of the extra features are remain undelivered. The customer on using the core product tells the developer upon certain changes that are required and they are made over the time.

Most of the times, the customer defines general objectives that are to be covered in the product but does not gives a brief about the functionalities and features of the product. Also, the developer can be unsure of the algorithm efficiency, OS adaptability or human-machine interaction. In such conditions, prototype paradigm is the best solution.

RAD or Rapid Application Development is a type of incremental model. In this, the project is developed in mini projects and then all of them are combined so that an efficient system can be built up in parts.

Agile process model emphasizes on project agility. It is relatively easier to make changes when the product is in requirements gathering phase. It is a fast process. It requires less documentation and the teammates coordinate quite well.

Over the past few years, many software projects have failed. Not only small companies, even the big companies like

Microsoft have failed to deliver successful projects. For a project to be claimed as successful, certain attributes should be considered.

- Clear Requirements
- Proper planning and monitoring
- Proper technology and management
- User involvement
- Dealing with change
- Stakeholder satisfaction
- Sound project management
- Skilled resources
- Modest Execution
- Clear Business Objective
- Emotional Maturity
- Optimization

Following these practices will improve the success rates of the project. But, even after following the best practices, many software still fail. The reason behind the failure of such software projects can be due to one or many of the following reason:

- Frequent changing requirements
- Use of the wrong methodology
- Use of wrong technology
- Unmanaged risks
- Badly defined requirements
- Commercial pressures
- Inability to deal with project's complexity
- Unrealistic project goals
- Poor communication among stakeholders
- Poor project management

The most referenced groups, the Standish Chaos [4] report, provided the statistics of software project failures. They have defined the success rate of software projects. The table for the same is shown below:

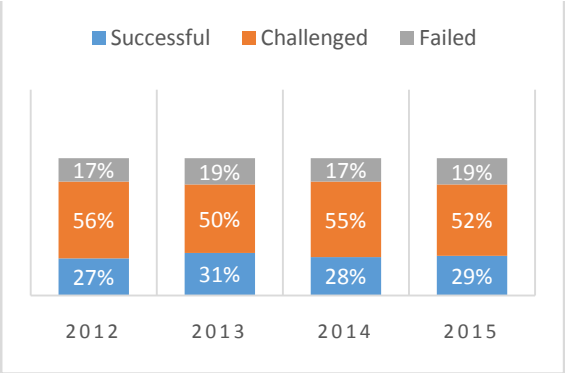

Figure 1. Figures for project status

Another survey which conducted by them shows that how the probability of success of a smaller project is much higher than the larger ones. This was conducted from FY 2011-15. The figures for the same are shown below:

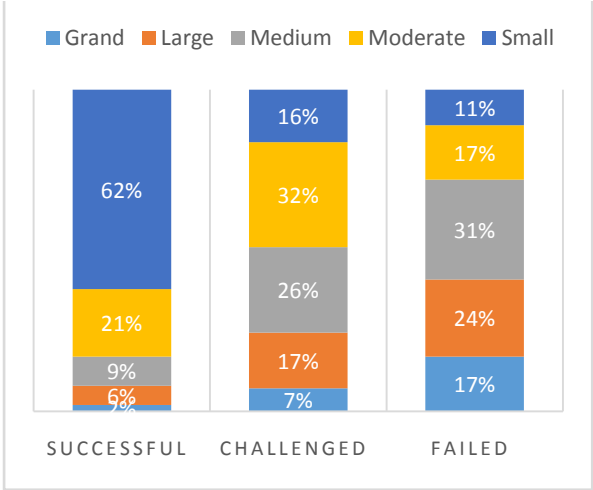

Figure 2. Statistics for project status depending upon size

The failures of the software projects in 2007 by TCS (Tata Consultancy Services) was reported as follows:

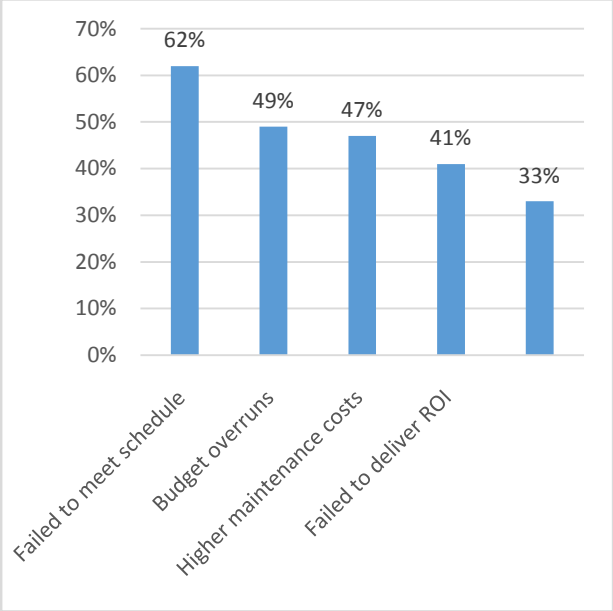

Figure 3. Reason for Failure

Another statistics of a research conducted by Oxford University [5] (Saur and Cuthbertson, 2003) regarding the IT projects success in 2003 is shown:

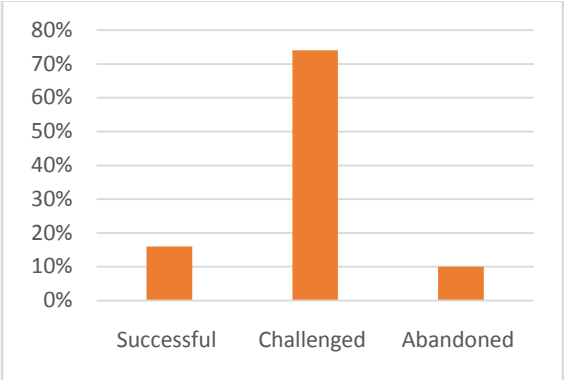

Figure 4. Oxford Univ. stats for IT Project

All this statistics emphasizes the need for developing and enhancing the software development process and that should be done especially for the challenging process as they are increasing. Most of the researchers have classified the methodologies into two. One is heavy weight methodology and the other is light weight methodology. This paper discusses both the methodologies and their constituents in detail.

II. HEAVY WEIGHT METHODOLOGY

Heavy weight methodology [6] consists of a sequential series of software development phases which include analysis, design, implementation and testing. It mainly contains long plan and design and documentation before starting the development. Some of the well-known models, with their advantages and disadvantages are presented below:

A. Waterfall Model

Waterfall model, also known as classic life cycle, is a well-known and the most basic software development heavy weight model developed in 1970. Although it's an old one and needs revision, it is still used for its simplicity. In this, next phase of development begins if and only if the previous phase of development is completed. It does not allow the process to go backward. In another word, it is unidirectional. It is used only in the case where the enhancements to the existing system is to be made or when the proper requirements are known in well advance. It acts as a baseline for future reference. The phases of waterfall model are communication, planning, modeling, construction, deployment. These are analogous to planning, analysis, design, implementation, testing and maintenance.

1) Advantages
- Full and proper documentation
- Well-known and basic model
- Easy to measure project status
- Full planning is done at the beginning of the project

2) Disadvantages
- Less user involvement
- Takes long time to deliver the software system
- High amount of risk
- Not good for big projects
- Requirements may not be clearly known, especially for applications not having existing (manual) counterpart
- It is assumed that the requirements do not change with time

B. Spiral Model

Spiral model is a combination of waterfall model and prototyping model. The phases used in the spiral model are in similar fashion to the one used in waterfall model in an iterative manner. This software development model was introduced in 1988 to remove the shortcomings of the waterfall model which includes changing requirements during the development. In this, a new version is released at the end of each iteration. The phases in his model are developed in a cycle known as a spiral. It is mainly used for large projects where risk involvements and requirements changing rates are high. Also, the whole project is decomposed into smaller projects. The number of loops are not fixed. It provides direct support for coping up with the project risks and also the risk analysis is done through prototype construction. After several iterations along the spiral, all the risks are resolved.

1) Advantages
- Product is delivered shortly
- Suitable for large and complex projects
- Changing requirements is possible during construction
- Risk assessment is also carried out at the end of each spiral
- High user involvement in comparison with the waterfall model
- Initiation of the project can take place without knowing the complete requirements.
- Functionalities can be added later on.

2) Disadvantages
- Cost expenses are high due to varying requirements
- Need an expert to deal with the changing requirements
- Not good for small projects
- Risk analysis is difficult

C. Incremenetal Model

The introduction of incremental model was aimed to overcome the shortcomings and problems in the waterfall model. The purpose of this model is to introduce a software system through a set of iterative cycles and with each cycle, we will get a portion of the system which is known as an increment.

This model aims at developing big systems by breaking it down to small ones and dealing with each of this segment as an independent project. This will reduce the risk involvement in our project. It is useful for product development where developers define scope and features to serve many customers. For example database products, payroll or accounting packages. Also, early version with limited feature important to establish market and get customer feedback. Consider an example. Suppose we want to develop a word processing-software using the incremental model. Then the first increment that would be delivered will include basic file management, document production and editing functions. Second increment will consist of more sophisticated document editing and production capabilities. The third increment could be spelling and grammar checking and advanced page layout can be the fourth increment. In this way, a software can be deployed using the incremental model.

1) Advantages
- Early problem discovery
- Any change in requirement can be handled between the increments
- Easier to test
- Quick delivery of the working system

2) Disadvantages
- Requires user involvement
- Expensive
- Heavy documentation required

III. LIGHT WEIGHT METHODOLOGY

Light weight methodology [7] gives developers chance to construct software rapidly and efficiently with less documentation and more responsive to the change in requirements. It lays emphasis on short cycles, encourages user involvement and delivers a working model at the end of the cycle in less time. Some of the well-known light weight methodologies for software development with its advantages and disadvantages are discussed below:

A. Prototyping model

Most of the times while developing a software, the customer mentions the general objective of the project keeping aside the detailed requirements, functions and features. In such cases, the developer might develop a prototype or model and in that model, the algorithms might not be appropriate and the OS efficiency might have been ignored. In total, only a working

model with a quick design is delivered in the beginning. Users use the model and ask for certain changes and then the cycle continues. If that prototype is a good start, then they carry on with the same model and make improvements and add further functionalities. In this, the initial model only focusses on how the software will be visible to the end users and after evaluating, the back end process starts. It involves various steps such as determining the basic requirements including desired input and output information, developing an initial prototype, reviewing the prototype by the user and then revising and enhancing it. In the initial stages, reliability and security concerns are ignored. By using the feedback from the user, the further model is developed based on the requirements. Negotiation is done then on the basis of the scope of the functions.

1) *Advantages*
   - Faster development
   - Early delivery
   - Cost saving
   - Easy to integrate with other models
   - Easy to refine and define the requirements
   - Demonstrate technical feasibility of the system
2) *Disadvantages*
   - Lack/ Poor documentation which makes it difficult to maintain
   - Poor quality due to fast development
   - Can lead to optimism

B. *RAD Model*

RAD Model or Rapid Application Development model is an iterative based-development which does not include any particular planning. It uses minimal planning for rapid prototyping. In this, the functional modules are developed independently in parallel and are integrated later on to complete the software product. Due to the absence of detailed preplanning, incorporating changes with development is easy. The main aspect of this is to make sure that the prototypes that are developed are reusable. The various phases of RAD model include business modelling, data modelling, process modelling, application generation and testing and turnover. It enables rapid delivery of the product. It requires availability of highly skilled engineers and the customer who committed towards the target. Lack of commitment on either side can lead to failure of this model.

1) *Advantages*
   - Reduces development time
   - Progress can be measured
   - Quick initial review occurs
   - Encourages customer feedback
   - Changing requirements can be accommodated
   - Increases reusability of components
2) *Disadvantages*
   - Requires high skilled developers
   - Management complexity is more
   - RAD built only system that can be modularized
   - High dependency on modelling skills
   - User involvement throughout the life cycle is must
   - Suitable for shorter time development projects

C. *Agile Development Model*

It is a type of increment model. Developing a software in increment and rapid cycles result in small release and with each release, functionalities are build up. Each release will be tested and software quality is assured. It is generally used when new changes need to be implemented. In this model, new changes can be implemented at a very low cost in comparison with the other models. For implementing a new features, the developers need to roll back for a few days and then implement it. It believes that every project needs to be handled uniquely and also the existing techniques should be changed to get the best fitting model for the existing software development model. The final build contains all the requirements mentioned by the customer. Continuous customer interaction is an important condition. Pair programming is encouraged. Working software or a prototype is considered as a best practice for a customer to see and inform the changes. It also focuses on quick response for change and development by the customer.

1) *Advantages*
   - Realistic approach to software development
   - Suitable for varying requirements
   - Easy to manage
   - Little/no planning required
   - Delivers early partial solution
   - Resource requirement are minimum
   - Promotes teamwork
2) *Disadvantages*
   - More risk involvement
   - Depends on customer interaction
   - New team member involvement is an issue
   - Unsuitable for handling complex dependencies
   - Practicing it is must
   - High individual dependency due to less documentation
   - Not good for large projects

IV. COMPARISON BETWEEN METHODOLOGIES

Heavy weight methodologies consist of documents, proper planning and large teams whereas light weight methodologies consist of less documentation, less planning and small teams with great collaboration [8]. They are usually small projects. There are some criteria which can be used to select a methodology for a certain type of project:

- *Documentation*: Light methodology mainly concentrates on working on project whereas heavy weight rely on proper documentation without starting the project. It depends upon the project that which approach it should follow. If the software needs to fix small bugs or make small changes, then however, documentation is not required. If we are building the project from scratch and working in a team, we require a proper documentation for synchronization.
- *Team size*: Team size is small in light methodologies whereas large team is required in heavy methodology. Light methodology usually covers small or short timed projects which involve less man power in comparison with heavy weight methodology.
- Planning: Pre planning is required in heavy weight which results in a proper documentation whereas the planning is informal in light weight and hence, no proper documentation is required.
- *Requirements*: Requirements are mentioned previously in heavy weight due to large project size. In case of light weight due to small project size, requirements, if not mentioned earlier, is not a big issue.
- *Communication:* Light methodology has face to face communication between developers and the customers. This helps in delivering a good software with all the users requirements fulfilled.
- *Change of requirement:* Heavy methodology does not accept the change in requirements and involves heavy costing at last phases. It is because if there is a change in the development phase, then there are many inter-dependent links that need to get updated for proper

working on the software. Light methodology validates the change of requirements.

V. FACTORS FOR CONSIDERING MODEL

We have seen several models which are used for software developments. Different models have different characteristics. They have their own features [9]. Below are the tables comparing light weight methodologies and the heavy weight methodologies software models:

Table I. For Heavy-weight Methodology

Properties	Waterfall	Incremental	Spiral
Planning in early stage	Yes	Yes	Yes
Handle Large projects	Not appropriate	Not appropriate	Appropriate
Cost	Low	Low	Expensive
Maintenance	Least	Promotes maintainability	Typical
Risk involvement	High	Low	Medium to high risk
Framework type	Linear	Linear + Iterative	Linear + Iterative
Team size	Large team	Not large team	Large team
Objective	High assurance	Rapid development	High assurance

Table II. For Light-weight Methodology

Properties	Prototype	RAD	Agile
Frequency of change	High	High	High
Documentation	Low	Low	Low
Delivery time	Early	Early	Early
User involvement	Required	Required	Required

VI. CONCLUSION

The variety of software models vary depending upon the factors of requirement of the project. Software models, when used appropriately, will give us better results. When an appropriate model is applied according to the requirements, the success rate of software projects can increase. There is no perfect model existing. Everything depends upon the project and its characteristics like nature of the project, developer's skills and management support. Many times, models can be combined together to get a different model which are known as hybrid model.

VII. REFERENCES

[1] Khan, A.I., Qurashi, R.J. and Khan, U.A., 2011. A comprehensive study of commonly practiced heavy and light weight software methodologies. arXiv preprint arXiv:1111.3001.

[2] Beeson, J.D. and Yates, J.B., Concurrent Ventures, LLC, 2016. System and method for dynamically load balancing across storage media devices having fast access rates. U.S. Patent 9,436,404.

[3] Kahtan, H., Bakar, N.A. and Nordin, R., 2012, October. Reviewing the challenges of security features in component based software development models. In E-Learning, E-Management and E-Services (IS3e), 2012 IEEE Symposium on (pp. 1-6). IEEE.

[4] Standish Group International Inc, chaos chronicles, 2016

[5] Galorath, D., 2008. Software Project Failure Costs Billions-Better Estimation & Planning Can Help. Project Management.

[6] Vanalle, R.M., Baptista, G.L. and Salles, J.A.A., 2015. A Software Development Process Model Integrated to a Performance Measurement System. IEEE Latin America Transactions, 13(3), pp.739-745.

[7] Blum, B.I., 1992. Software engineering: a holistic view. Oxford University Press, Inc..

[8] Ben-Zahia, M.A. and Jaluta, I., 2014, June. Criteria for selecting software development models. In Computer & Information Technology (GSCIT), 2014 Global Summit on (pp. 1-6). IEEE.

[9] Faradani, H., 2011. A Guide to selecting software development methodologies.

[10] Spector, A.Z., 1990, April. Achieving application requirements. In Distributed Systems (pp. 19-33). ACM.