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.
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:

<table>
<thead>
<tr>
<th>Year</th>
<th>Successful</th>
<th>Challenged</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>17%</td>
<td>56%</td>
<td>27%</td>
</tr>
<tr>
<td>2013</td>
<td>19%</td>
<td>50%</td>
<td>31%</td>
</tr>
<tr>
<td>2014</td>
<td>17%</td>
<td>55%</td>
<td>28%</td>
</tr>
<tr>
<td>2015</td>
<td>19%</td>
<td>52%</td>
<td>29%</td>
</tr>
</tbody>
</table>

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:

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

<table>
<thead>
<tr>
<th>Reason for Failure</th>
<th>2012</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failed to meet schedule</td>
<td>62%</td>
<td>49%</td>
<td>47%</td>
<td>41%</td>
</tr>
<tr>
<td>Budget overruns</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Higher maintenance costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failed to deliver ROI</td>
<td>33%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

Table II. For Light-weight Methodology

<table>
<thead>
<tr>
<th>Properties</th>
<th>Prototype</th>
<th>RAD</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency of change</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Documentation</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

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