INTRODUCTION

Software architecture is the high level blueprint of the system to be built. The production of quality software depends on implementing the best plan by applying engineering. This chapter starts with the background of this research by giving a brief introduction about the engineering and software engineering concepts. It displays the evolution of the software development life cycle and explains its various stages. Software development process models and their history are presented here. The research problem and objectives are defined elaborately. Further, the scope and limitations of this research are discussed in this chapter.

1.1 RESEARCH BACKGROUND

Engineering is the discipline that stands between the invention and economy to deliver the quality product in the market. It applies the methods and processes to make the product to reach the top of the market. Similarly, software engineering is the process of applying engineering techniques into the software development to produce the quality software with optimal cost. A definition proposed by Fritz Bauer reflects the same meaning: “Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines [1]. “ Bauer’s definition provides us with a baseline of cost effective quality software. Sound engineering principles, Economic software and efficient software are the challenges to software engineers. Institute Of Electrical And Electronics Engineers (IEEE) defines Software Engineering as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software [2].
Software is the overriding component regarding the expense and multifaceted nature. Good software engineering practices and tools can in this manner have a generous effect, even to the degree that they may be the main impetus of the undertaking success. Software engineering is performed by creative, knowledgeable people who should work within a defined and matured software process. This process follows Software Development Life cycle. The main objective of this life cycle is to produce a high quality software that satisfies the customer’s requirements, cost and time constraint. The next section explains the various stages of this life cycle and its evolution.

1.2 SOFTWARE DEVELOPMENT LIFE CYCLE

To solve a given problem or to automate a procedure for a software development, code developed by the programmer was just enough during 1940’s and 1950’s [3] [4]. But when the complexity of the problem is to be automated and increased, there was a need to break down the complexity to solve the problem. This made a way for the design phase to be included along with the coding in 1960’s [3] [4]. In Late 1960’s, when the compulsion to meet business and consumer needs arose, there needed a mechanism to record their interests and to plan according to it. This made the analysis phase as the part of the software development process.

In order to ensure the quality of the product and to verify whether the product met the specified requirements, testing was introduced later in 1979 instead of debugging [5]. Even though “software maintenance” was proposed from the 1960s, it came into common use few years later. Due to Canning’s paper of 1972 and Swanson’s typology of maintenance activities introduced in 1976 made the acceptance of the “Maintenance” in the software development [6] [7] [8]. Under these circumstances the term “Software Development Life Cycle” was born.
Software Development Life cycle (SDLC) is a process that describes the life of a software product as a series of stages from the product’s conception to retirement [9] [10] [11]. The stages of SDLC basically include Analysis, Design, Development, Testing, Maintenance, Support and finally retirement.

Analysis is the most important phase in SDLC. It involves the elicitation of the system’s functional and nonfunctional requirements, and the specification of the system. Functional requirements deal with the function of the system. The Nonfunctional requirements deal with the behavior and quality of the system. Changes made in the nonfunctional requirements will have a severe impact on the production of the software. So, it is very important to capture the nonfunctional requirements and their priorities. At the onset, the customers may be uncertain about the function of the product they required and they may not fathom the end client’s needs. There may additionally be disagreements. If the requirements elicitation stage fails to capture the requirements clearly, then it will have the major impact on producing the quality software. For example, changes in priority from one Functional requirement to another requirement may have 100% impact on the software being developed. Hence, Senior members of the team will perform the analysis on inputs from different stakeholders.

The basic project approach and the product feasibility study in the economic, operational, and technical areas are planned based on the information derived from the analysis. Quality assurance requirements and risks associated with the project are also identified in this stage. The product’s functional and nonfunctional requirements are well defined and documented as a Software Requirement Specification document (SRS) after the requirement analysis. And, SRS is finally approved by the right stakeholders.
The next phase, Design includes the architecture and algorithms for the implementation of the product. SRS is the reference for Software architects to come out with the right architecture for the product to be developed. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins [12]. The suitable design approaches are proposed based on the requirements specified in the SRS and they are documented in a Design Document Specification (DDS).

DDS are reviewed by all the important stakeholders and the best design approach is selected based on various parameters like risk assessment, design modularity, budget and time constraints. The interconnection between the components, communication and data flow representations are clearly defined in a design approach.

The Development Phase is the key step in the project. The previous phases pave the foundation for system development. The system design prototyped in the Design Phase is converted into a working information system that meets all system requirements. The implementation of software design begins in terms of composing system code in the suitable programming dialect and creating executable projects effectively. In addition, the system should be verified and validated. The working system will now enter into the testing phase.

The system is tested against the requirements to verify that the product meets the needs specified during the requirements phase. Testing resolves the errors and faults in the software application. IEEE defines Error as “A difference between a computed result and the correct result” and Fault as “An incorrect step, process, or data definition in a computer program”, that means the error is the result of fault. These errors may be in the unit level or system level. After iterations of testing, the application will be free of errors that disturb the application.
Testing techniques and tools are needed to perform testing at various levels like unit and integration testing. Product’s defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS. The tested software will meet the business perspectives. It is released formally in the appropriate market. The product may be released in a limited segment and will be tested in the real business environment. Then, based on the suggestion for enhancements, the product may be released with the additional features in the market. After the release of the product in the market, its product maintenance phase starts.

The maintenance phase involves performing changes, corrections, additions, and upgrades to ensure the system continues to meet its business goals. It provides a help desk to support the system users.

Various Industries have tried to create and standardize the SDLC process within their industries. This makes the possibility to improve program management and technical training. This also makes the staffs feeling easily able to move from one project to another. Instead of concentrating on the SDLC process design issues, the technical and Management team can focus on the system design and development issues. As every organization has its own needs and infrastructures, slightly different SDLC’s can be used. This is done by following the suitable SDLC model for their organization. Various SDLC models with their strengths and demerits along with their evolution are discussed in the next section.

1.3 SDLC MODELS

Different types of activities are carried out during each life cycle phase and before the end of the phase, necessary artifacts should be produced. The various phases of the software life cycle, and the sequence in which those phases are executed are described in Software life cycle models. The Software
life cycle model maps the different activities performed on a software product from its conception to retirement into a set of life cycle phases. Different life cycle models may map the basic development activities to phases in different ways [13].

Widely Known and used of these, is the waterfall model. This model has a sequence of stages in which the output of each stage becomes the input for the next. Even though this model is widely used in industries, Royce was the first to write about this model in 1970 [14]. This waterfall model assumes that the only role for users is in specifying requirements, and that all requirements can be specified in advance. Unfortunately, requirements grow and change throughout the process and beyond, calling for considerable feedback and iterative consultation. Thus, many other SDLC models have been developed.

The incremental and iterative models are variants of the waterfall model. When the size of the project size gets increased, it will be easier to handle by dividing the project into smaller components and developing them incrementally or iteratively. Harlan Mills, who worked at the International Business Machines (IBM) Federal Systems Division (FSD) promotes iterative development in his well-known paper "Top-down Programming in Large Systems" in the year 1970 [15]. This was published in Debugging Techniques in Large Systems. In the iterative model, each component followed a waterfall model, passing through each step iteratively. Later in 1975, Williams introduced the incremental model in the International Conference on Reliable Software [16]. The components were developed in an overlapping fashion in the incremental model.

In 1988, Boehm developed the spiral model as an iterative model which includes risk analysis and risk management. James Martin published a book in 1991 about his rapid application development approach during at IBM during
1980’s. In the rapid prototyping (sometimes called rapid application development) model, the initial emphasis is on creating a prototype that looks and acts like the desired product in order to test its usefulness [17]. The prototype is an essential part of the requirements determination phase, and may be created using tools different from those used for the final product. Once the prototype is approved, it is discarded and the "real" software is written. This model was proposed in 1988 by Luqui [18]. The evolution of software life cycle and life cycle models are depicted in the following fig 1.1.

These models can be grouped under heavyweight or predictive and lightweight or adaptive approaches. The models developing the software in a traditional way of using a sequence of series of steps come under the heavy weight or predictive approach. The stable set of requirements should be
defined and documented at the beginning of the development life cycle in this approach. Examples are waterfall model, incremental model, spiral model, etc. The models that apply short development iterations throughout the project life-cycle fall under light weight or adaptive approach. Examples are the agile model, prototype model, RAD model and etc.

**The Waterfall Life Cycle Model**

The first formal description of the waterfall model is cited by Winston W. Royce8, The waterfall model is the classical model of the software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model gives emphasis to planning in beginning stages, it guarantees design flaws before they develop. In addition, its intensive document and planning make it work good for projects in which quality control is a major aspect. The waterfall development model originates in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

Waterfall model of software development is separated into many separate phases and follows a sequential software development process. In a waterfall approach different phases are interrelated and come in a linear style. Each phase suggests a different task, which is completely dependent on previous phase. The pure waterfall lifecycle contains several non overlapping stages, as shown in the following figure. The model starts with establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance. End of the one phase is the beginning of the other phase. Some certification mechanism has to be implemented at the end of the each phase. This is done
by some verification and validation. Verification means whether the phase satisfies the conditions imposed at the start of that phase. Validation means confirming the output of a phase is consistent with its input and the output of the phase is consistent with the overall requirements of the system.

Waterfall approach was the first SDLC Model used widely in government projects and in many industries. As it depicts the software development process in a linear sequential flow, it is called as a linear-sequential life cycle model. The outcome of one phase is the input for the next phase sequentially as shown in fig 1.2. Hence, it ensures that any phase can begin once the previous phase is complete. The pure waterfall lifecycle contains several non overlapping stages. All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases [12].

![Waterfall model](image)

Fig.1.2 Waterfall model
Each phase performs a different task that depends on the previous phase. This model is sometimes called the classical model as other models borrow from it and improve upon it. As this model prefers the planning in beginning stages, the design flaws are eliminated. This model is suitable for projects in which quality control is a major aspect. This model puts a definite entry and exit points of a project. Mistakes and conflicts are detected earlier in its each stage. This model provides the product deadline control as each phase of its process depends on the previous phase. This model is very efficient when team members are in different locations. When the product is the new version of an existing one or porting the existing product to a new platform, the waterfall model is the preferred model to implement. This model performs well for a product with a stable product definition and well understood technical methodologies.

As there is no way to go back, it is completely locked after each stage. The whole process should be started again to make any changes in the system. The variation of this model, modified waterfall model allows us to go back and redo our actions as shown in fig 1.3. But it doesn’t allow forward jump from one phase to another phase by skipping the intermediate phases.
Incremental Model

The Incremental model constructs a partial implementation of a total system and then the increased functionality is slowly added. This model prioritizes requirements of the system and then implements them in groups. The basic requirements are initially addressed in order to reduce the complexity and scope of each phase in a life cycle, and released as a core product. The next level prioritized requirements are implemented a part of the system, through several iterations of development life cycle. The integration of the increments results in the final system. Every release of the system adds function to the previous release, until all designed functionality has been implemented. The other features are added subsequently after the evaluation of the core product. As shown in below fig. 1.4, the increments represent development that can be carried out independently and in parallel. This model
is particularly useful when staffing is limited for a complete implementation within the deadline of the project [12].

![Incremental Model](image)

**Fig.1.4 Incremental Model**

Few people may work for the initial implementation. Further staffs may be added if the core product is well received. The risk of system failure will be less in this model rather than other models. Milestones can be set to measure the progress. Testing will be easy. This model is very useful to deliver the product quickly to the market with core functionality. When new technology is introduced in the project, the user will be comfortable with the small incremental part of system rather than the whole big system. This is applicable for low to average risk project.

**Iterative Model**

The reason of iterative developments is to allow reworking on part of a system to remove mistakes or to make improvements based on user feedback. The iterative model is shown in fig 1.5.
The number of iterations that would need to occur could be uncertain depending on how much of the system would need to be corrected; so, it may be difficult to set an end to the project. But it can go as long as required system is not developed. Iterative model repeats every step after every cycle of SDLC process in a cyclic manner as shown in fig 1.5. After iteration, an evaluation of the result and planning to improve the correctness of the system and the quality of design would need to happen.

Initially, small scale software is developed by following all the phases of SDLC. Extra features and modules are designed, coded, tested and added to the software in the next iteration. Risk management can be worked out after
the iteration. The iterative model can be used with the models like incremental model and water fall model for improved software development life cycle.

**Spiral Model**

Waterfall model is the simplest models of software development process. Carrying the errors to the next phase makes this model, to take more time to solve the problem. This risk will affect the success rate of the software product. In order to overcome this problem, the spiral model was designed to mitigate risk factor. This model is implemented in high risk projects. Spiral model is a combination of both, iterative prototype model and waterfall model [19]. The spiral model can be applied throughout the life of the software even after delivery [12]. Spiral model is also called as meta-model because it consists of other models of SDLC.

There are four phases in this model as depicted in fig 1.6. They are Planning, Risk Analysis, Engineering and Evaluation. These phases are iteratively followed to solve the problems in waterfall model.

The planning phase determines the goals, alternatives and constraints of the project and documents them. The SDLC model to be used during the life cycle is decided in this phase. This phase is considered as the core of the spiral model, so it requires more time and care.

The Risk Analysis phase is the special part of this model which could not be found in other models. Reducing risks is the main focus of this phase. All the possible risks are identified and solved in this phase. In order to build the cost-effective quality system, a deep analysis on available options has to be done. Risks may indicate any sort of instability in requirements. Then, the solution for the changes in requirements can be obtained by using the prototype with the available data. Operational and technical issues are also discussed here.
An Engineering phase deals with the coding part of the software development. Software built in this phase will go through all the phases iteratively to obtain improvements in the software and the system will undergo testing and fixing the bugs. The bug free software will be the outcome of this phase.

In the Customer Evaluation phase, developed product is released to the customer for evaluation, and issues identified by them will be resolved in the next prototype. Similar phases will be passed through in the subsequent loops of spiral model. Highly skilled people are required to develop software using this model. This is one of the most flexible models. The project manager will decide the development phases based on the project’s complexity. As project monitoring is done at each phase of each iteration, it makes this model more powerful to create the quality software. Implementing software using this model consumes high cost. It is hard to follow the rules and protocols that lead to the successful implementation of this model. As development progresses
iteratively, meeting the deadline and budget is somehow tough. Evaluating the risks and uncertainties need skilled people.

The spiral model is suggested for building the large system with more features. This model can be used when the user is unable to specify requirements clearly and trying new skills on the new technology. If the organization is not ready to invest all the money initially for the project, this model is the best choice.

**Prototype model**

When the customer is able to define objectives but unable to identify detailed input, processing and output, the Prototype model will be the best approach. This model allows the interaction of the customer to experience the working representation of the product. This model is useful when there is no existing system to find out the requirements. This model is used to develop completely a new system when there is no existing system or part of the system of same type is available. The prototyping model begins with requirements gathering as shown in fig 1.7. The overall objectives for the software is defined in the developer and customer meet.

A prototype is built based on the currently known requirements to capture all the requirements clearly, before finalizing the requirements. This prototype development goes through all the phases of waterfall model. The prototype gives better idea about the system’s requirements to the customer and is used to forward the feedback of the customer to the developer. The prototype is modified by iteration in order to satisfy the needs of the customer. So, the developer will have better understanding of customer’s requirements [12].

After getting the feedback of completely built sample model, the SRS document is prepared. Based on this document, the final product is developed
following the waterfall model. This model is mainly used in decision support system and user interface systems.
**Agile model**

Agile model is a combination of iterative and incremental process models. This model supports the requirement of stakeholder throughout the life cycle of the software development. This model provides a fast delivery of working software, focusing on the adaptability and customer satisfaction. Agile methods break up the problem and build the product by small incremental builds. These builds are provided in iteration as shown in fig 1.8. Each iteration involves cross functional teams working on different phases. Within a short period of time, the first working model will be delivered. At the end of the iteration, the product is displayed to the customer and important stakeholders. Iterations are of fixed length and therefore are said as time-boxed. Multiple iterations are required to release a product with new features. During iteration, the phases of SDLC will occur in parallel, such as requirements, coding and testing.

Fig.1.8 Agile model
The team contains a customer representative appointed by stakeholders to answer problem domain questions on behalf of stakeholders. Customer representative and Stakeholders will meet at the end of iteration to review the progress and reevaluate priorities to get maximum return on investment. This model focuses on face to face communication daily with the team members and customer representatives when they are located in same place to discuss about the written documents. They may contact daily through mail, voice chat, video conferencing etc, when they are in distinctive locations.

**RAD model**

Rapid application development (RAD) is an agile process model with an extremely short development cycle. This model requires minimal planning for development. The planning is interleaved with coding part of the software development. The lack of planning is allowed here in this model and faster coding of software makes this model to change the requirements easily. For the faster development of software, many tools that offer reusable components are available. As the components are fully functional by themselves, testing is very easy.

User’s requirements are classified and the final system is designed using structured techniques and prototyping of this model. The software development starts with the development of preliminary data model and business process model using structured techniques. In the next stage, prototypes are used to verify the requirements as shown in fig 1.9. This improves the data and process models. These stages are repeated iteratively, that results in new system to be ready for use.
This model is used when there are uncertain requirements, user’s obscure view about the system. This model is appropriate for the small scope projects and not for all projects. This model will work better when the task can be broken up into manageable units. This model is used to build a new system, with minor changes in the already developed components in different context.

To deal the changes coming from the business, there are no sufficient tools and techniques for this model. If the scope, functions, data, applications, output and process of the modules are clearly defined, then this model is more suitable. This model is very effective when the architecture and technology of the modules are of same type, since there will be a smooth integration. As this model focuses on faster development of a system, it may compromise in functionality and performance of the software.

The following table 1.1 is a comparison table of various models discussed above. This table clearly shows the model to be chosen based on the project needs. Based on the impact of quality needs, the appropriate model can be selected.
Table 1.1 Comparison of SDLC models [21] [22]

<table>
<thead>
<tr>
<th>Features</th>
<th>Waterfall</th>
<th>Incremental</th>
<th>Iterative</th>
<th>Spiral</th>
<th>Prototype</th>
<th>Agile</th>
<th>RAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Specification</td>
<td>Beginning</td>
<td>Beginning</td>
<td>Beginning</td>
<td>Beginning</td>
<td>Frequently Changed</td>
<td>Frequently Changed</td>
<td>Time boxed Release</td>
</tr>
<tr>
<td>Understanding Requirements</td>
<td>Well understood</td>
<td>Not Well understood</td>
<td>Not Well understood</td>
<td>Well understood</td>
<td>Not Well understood</td>
<td>Well</td>
<td>Not Well understood</td>
</tr>
<tr>
<td>Overlapping Phases</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>No</td>
</tr>
<tr>
<td>User Involvement in all phases of SDLC</td>
<td>At beginning</td>
<td>Intermediate</td>
<td>Intermediate</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Changes Incorporated</td>
<td>Difficult</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Easy</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
<tr>
<td>Cost</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Expensive</td>
<td>High</td>
<td>Very high</td>
<td>Low</td>
</tr>
<tr>
<td>Availability of reusable component</td>
<td>No</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Complexity of system</td>
<td>simple</td>
<td>Intermediate</td>
<td>simple</td>
<td>complex</td>
<td>complex</td>
<td>complex</td>
<td>very simple</td>
</tr>
<tr>
<td>Risk Involvement</td>
<td>High</td>
<td>Easily manage</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Reduced</td>
<td>Low</td>
</tr>
<tr>
<td>Guarantee of Success</td>
<td>Less</td>
<td>High</td>
<td>High</td>
<td>Good</td>
<td>Very high</td>
<td>Good</td>
<td></td>
</tr>
<tr>
<td>Implementation time</td>
<td>Long</td>
<td>Very Long</td>
<td>Short</td>
<td>Long</td>
<td>Short</td>
<td>Very less</td>
<td>Short</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Rigid</td>
<td>Less Flexible</td>
<td>Less Flexible</td>
<td>Flexible</td>
<td>Highly Flexible</td>
<td>Highly Flexible</td>
<td>Highly Flexible</td>
</tr>
<tr>
<td>Expertise Required</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>Very high</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Cost Control</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Resource Control</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

These models discussed above cannot be used directly in the organization. In order to make the models to be effective, models should be used with software process like CMM, ISO, Clean room software engineering, etc. They aim to improve the software quality by improving the software development process. The next two sections explain the initial stages of the SDLC and the impact of nonfunctional requirements in these stages that are related to this research.

1.4 SOFTWARE REQUIREMENTS SPECIFICATION

Requirement specification is the document that consists of functional and non functional requirements with other requirements, produced at the end of the requirements elicitation and requirements analysis. It is used as a communication medium between the stakeholders. Once the software requirement specification is accepted by all stakeholders, the end of the phase has been reached. Even after its acceptance, the requirements can be changed,
but the changes will be tightly controlled. The software requirement specification should be edited by both the customer and the supplier, as initially neither has both the knowledge of what is required (the supplier) and what is feasible (the customer). The software requirement specification is used to capture the users’ requirements and if any, highlight any inconsistencies and conflicting requirements and define system and acceptance testing activities. A software requirements specification should be complete, traceable, verifiable and unambiguous. It must correctly specify all software requirements, but not the design or verification aspects.

*Software requirement* has been defined by IEEE [2] as, (1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document; (3) a documented representation of a condition or capability as in (1) or (2). Few requirements are believed to be implemented by the stakeholders. They are ‘known requirements’. Requirements that may be forgotten by the stakeholder as they are not required currently are known as ‘Unknown requirements’. The inability of the stakeholder to identify new requirements may arise due to the lack of domain knowledge. These requirements are ‘undreamt requirements’. These three type of requirements may be functional or non functional.

Performance, Reliability, Availability, Usability and Flexibility are the most important non functional quality requirements for the user. For the acceptance of the product, it is essential to meet the non functional requirements as they have direct relation to customer’s expectations. The customer’s satisfaction and intension to buy the product rely on meeting the non functional requirements. The non functional requirements play vital role to achieve the success of the product. They have serious impacts when design
is carried out. These impacts will reflect on the functional requirements also. Hence, it is very important to capture the Non functional requirements rightly, prioritize and balance them.

Requirements elicitation

Elicitation is the determination of the stakeholders’ needs and finding out the needs of users and stakeholders by understanding and extracting their needs [23]. Requirement elicitation is recognized as one of the most critical knowledge intensive activities of the development of software. Studies by [24] indicate that 70% of the system errors are due to the inadequate system specification and 30% of the system errors are due to design issue. There are different ways to get the required information and approach problem. One is direct approach, another is indirect approach. First one classifies the methods by whom we interact with the domain expert and second one classifies them by what type of information is obtained.

Direct Approach: To improve the understanding of the problems of a current system, the direct approach is used. Most common techniques used are Interviews, case study, Prototyping. A complete analysis of the total procedure can be obtained using these tools. The right data and more knowledge about the system can also be obtained. To achieve these, the domain expert should assist to share information. [25].

Indirect Approach: To acquire the information that can't be effectively enunciated, indirect methods are used. Questionnaires, documents analysis are its examples. By analyzing the documents, a bulk of data can be obtained. The results acquire from this type of investigation are easy to measure and an applicable test suggestion can be driven from them [25].
Analysis of the Gathered Requirements

After gathering all the requirements, the gathered requirements are analysed to understand the user requirements. Detect inconsistencies, ambiguities, and incompleteness. Incompleteness and inconsistencies can be resolved through further discussions with the end-users and the customers. Inconsistent requirements are some part of the requirement that contradicts with some other part. Incomplete requirements are some requirements that have been omitted due to oversight. Requirements analysis involves obtaining a clear, in-depth understanding of the product to be developed. It eliminates all ambiguities and inconsistencies from the initial customer perception of the problem.

Due to the absence of the working model of the problem, it is not easy to get an in-depth understanding of the problem. Even though it is tough task, experienced analysts will spend considerable time with customers to understand their exact requirements, to develop a satisfactory system. Then based on their understanding, a building model or prototype is built to understand whether they have really captured the right requirements. The next section explains how the requirements are converted into design.

1.5 DESIGN

Design converts the Software requirement specification into the skeleton of the product that is partially implementable. Good design reduces the implementation effort and cost. The Software requirement specification will be analyzed by the design engineers.

System design is the phase where system engineers analyze and understand the business of the proposed system by studying the user requirements document. They plan our possibilities and techniques by which the user requirements can be implemented. If any of the requirements are not
feasible, the user is informed about this issue. A resolution is found and the user requirements document is edited accordingly. Designs are at two levels. The high level design deals with the overall design of the system also known as system architecture or software architecture simply. The low level design deals with the detailed implementation design.

The High Level Design phase focuses on function allocation, domain understanding, managing stakeholder expectations and creating the test strategy. In this phase the architecture is established. The design of the complete system is outlined briefly in this phase. This phase determines the software framework or architectural style of a system to meet the specific requirements. Thorough knowledge of the requirements is very important to come out with good high level design.

As the non functional requirements in the SRS have major impact on the design, Software architect should carefully analyze and understand their priorities. Selecting the suitable architectural style based on the non functional requirements and other requirements, is the crucial task for the software architect. This selected appropriate architectural style defines the major components and the interaction of those components, but it does not define the structure of each component. This will act as the blueprint of the system.

The frozen requirements cannot be easily manipulated as they come to the design phase, because the cost involved in the requirement elicitation process is 10% of the project cost [26]. Especially, changing non functional requirements will have major effect on the total project cost. If a heavy push on changing the requirements or changing the requirements priority arises, after the preparation of architectural plan but before the implementation, then a new mechanism to handle this situation should be implemented. The research in this dissertation focuses on this problem in order to ameliorate the performance of the software architecture to engineer the quality product. The
trade off analysis of complexity makes many things simple and makes the way to high quality product. The scenarios are converted into a test plan by the architecture team.

Low level or detailed design will be the designer’s next focus. Every module's functionality will be determined and defined. Constraints on the use of its interface, pre and post conditions, and module-wide invariants are specified and internal data structures and algorithms are also suggested. Design phase represents the "how" phase of the software development lifecycle.

1.6 PROBLEM STATEMENT

Software Architecture is a high level blueprint of the system to be built. Creating architecture during the software development is a critical task and an appropriate technique for selecting a right architectural style is also very important, as there exist many architectural styles with different ratio of quality attributes. Quality attributes derived from the requirements elicitation play vital role in selecting the best architectural style. Quality attributes are relative to each other. They will affect each other positively or negatively when their value gets changed. It is not possible to enforce the quality attributes during development and evaluate for the customer prior to delivery. And also it is difficult to ensure that the selected architecture style exactly matches, with the user preferred right proportionate of quality attributes that meets the needs of all the stakeholders involved in software development and usage, known as balanced quality attributes.

There is no model or technique in literature to verify the balanced quality attributes during the selection of the software architecture style, known as architectural analysis phase. Hence, picking up the right architecture style with a balanced quality attributes to create the architecture based on the user
and quality requirements become the complex activity. Also the selection of architectural style is static in nature. Even if any changes are required based on the stakeholder’s requirements, then again the whole process of architecture evaluation has to be restarted. This may again take few more months and consume major chunk of project cost to bring out the architectural style to be implemented. Hence, there is a need to analyze the architectural style dynamically and to freeze before it goes for the implementation. Currently used techniques in the literature like Software Architecture Analysis Method (SAAM), Architecture Tradeoff Analysis Method (ATAM), SAAM for Evolution and Reusability (SAAMER), Cost Benefit Analysis Method (CBAM), Scenario based Software Architecture Reengineering (SSAR), ArchDesigner, WIN CBAM and Non Functional Requirement Framework (NFR Framework) are time consuming and are heuristically validated. They are based on brainstorming session and once the architectural decision has been taken, it is difficult and costly to change the decision. Selection of the architectural style is static in nature. The exact user preferred composition of quality attributes are not taken into account for the architectural style selection. Hence, this research work solves these issues by mapping the Vastu mandalas to existing software architectural styles and by proposing a new model for the dynamic selection of a right architectural style with balanced quality attributes.

1.7 RESEARCH OBJECTIVES

- To study vastu indepth to mine the available patterns in vastu literature.
- To make a comparative analysis on software architectural styles with vastu patterns
- To derive a model based on the study to dynamically select the architectural style.
1.8 SCOPE AND LIMITATIONS

SCOPE

The Scope of this research work is limited to the Software architecture in the design phase of the SDLC. This research work intends to assist the software architect to select the right architectural style when the requirements get changed after fixing the architectural style for the implementation.

LIMITATIONS

In this research, Security is taken as one of the quality attribute and not considered in detail from software development perspective like Data security, Network security and etc. If the weightage is to be given to the stakeholder’s views during validation, then geometric mean will not be suitable for prioritizing quality attributes. Risk mitigation is not built in the hybrid model taken into account in this research.

If risks like technology risk, SRS risk, Business impact risk and Development environmental risk are taken into account, then the proposed model needs to be modified with additional attributes. Only sixteen quality attributes are considered in this research work. Additional quality attributes may be considered by expanding the mapping table.

1.9 ORGANIZATION OF THE THESIS

This dissertation is organized as follows.

Chapter 1 provides the background information of the research work. The research problem and research objectives are defined. Scope and limitations of this research is discussed.

Chapter 2 deals with literature review. It briefs the evolution of software architecture and discusses the Software architectural styles and frameworks.
The detailed information of vastu and its patterns are also presented in this chapter.

Chapter 3 describes the quality attributes and various quality models.

Chapter 4 presents a comparative study of Architecture analysis methods.

Chapter 5 explains the proposed hybrid model.

Chapter 6 shows the experimental validation of this research work with a Meeting scheduler case study.

Chapter 7 presents the conclusion drawn from this research and lists the possible directions of future researches.