CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The role of software has been increasing in our life day by day. Earlier it was limited to desktops only, but now has reached to the devices that can easily accommodate in our pockets. Nobody can think about a life without the devices controlled by software (Dalal et al., 2014). Such dependence as well as trust on software compels the software industry to be more conscious and attentive while developing software, so that delivered software became successful in their operational life (Rizvi, et al., 2016a). On the other hand, it has also been noticed that, in industry most of the development activity is carried out in labor-intensive manner (Pandey and Goyal, 2013). System developers are also struggling to deliver software with acceptable level of quality, within given resources and time. Such a pressure on the software professionals cannot be ignored as one of the key factor for software whose reliability is not up to mark (Hall et al., 2012).

A lot of unfortunate events had already occurred in the defense and health sectors due to the unreliability of corresponding software applications (Gerard, 1997;

Ogheneovo, 2014). After realizing reliability as one the key quality attribute, its prediction cannot be delayed or ignored. Therefore, there is an emergent need to ensure reliability of developing software as early as possible. So that developers can take suitable corrective measure before they start writing the actual code. In the last two decades, a large number of models for predicting the reliability have been proposed. But still, this domain of software engineering has been attracting more researchers to contribute further. It is evident from the review of the literature that reliability has been estimated or predicted through a variety of techniques like, formal methods, neural network, cause-effect graph analysis, multiple linear regressions, fuzzy logic, and many more. But prediction at early stage was rarely discussed (Yadav et al., 2012).

The researchers had put their best efforts, but still there are a number of theoretical and practical issues noticed in many studies, that undermine the strength as well as validity of most models. Appropriate validation process is a necessity for the success of every effort. But majority of the models lack quality validation. Even though, it is a universally accepted statistic that 70 - 80% of all the faults in software are get introduced during the requirements phase, this phase of the SDLC had not been given importance while quantifying the reliability. Majority of existing reliability models are applicable only in the later stages of development, and helping developers either by the end of coding phase or in the testing stage (Yadav and Yadav, 2014). That becomes too late for developers to take corrective measure to improve its reliability.

Despite the obvious variety of software reliability quantification, a prospect to design and implement a comprehensive framework that can be followed by industry personnel and researchers to quantify reliability on the basis of requirement and design stage measures appears highly advantageous and significant. This dissertation addresses the aforementioned need of early estimation of reliability.

1.2 SOFTWRAE QUALITY

Software seems to be affecting every aspect of human life. Software quality is no longer an advantage but a necessary factor for software industry as it may cause for human life, time and budget. The demand for quality software continues to increase due to our society's increasing dependence on software and the often-devastating effect that a software error can have in terms of life, financial loss or time delays (Kitchenham and Pfleeger, 1996). Embedded software systems must ensure the consistent and error free operation every time they are used. This demand for increased software quality has resulted in quality being more of a differentiator between products than it ever has been before. One of the major factors driving any production discipline is quality (Bansiya and Devis, 2002; Ma, et. al, 2012).

A major problem in quality engineering and management is that the term 'quality' is ambiguously defined, as it is commonly misunderstood. This confusion may be due to the fact that quality is a multidimensional concept rather than a single idea. About half of the software problems are reportedly caused at the requirements specification level, about 25% or more are caused at the design level and just a few are caused during coding (Basili et. al, 1996). The main goal of software engineering is to produce high quality software.

The concept of software quality can be described from five different perspectives (Booch, 1994): transcendental view, user view, manufacturing view, value view and product view (He, et. al, 2012). User view is the most common it defines software quality as ‘fitness for purpose’. The other references include ‘conformance to user requirement ’and ‘fitness for use’. These two references to quality are interrelated and consistent with each other. Conformance to requirement implies that software requirements must be clearly stated such that they cannot be misunderstood (Bansiya and Devis, 2002). Fitness for use reference takes customers requirements and expectations into account.

As far as the other four perspectives are concerned, the transcendental view describes quality as something that can be recognized but not defined, while the value based view describes quality in relation to the value that the customer is willing to pay for the software. Software engineers consider quality as conformance to specifications giving a manufacturing view of software quality and specific product characteristics are referred as the product view of software quality (Bluemke, 2001).

In order to quantitatively and qualitatively evaluate the software quality, various metric based quality estimation models have been established, most of which are hierarchical models. They are based on group of quality criteria, associated with group of metrics. Software quality models can be categorized into three kinds according to the means by which they are generated (Booch, 1994); the first is the theoretical model, based on relations among variables. The second is data-driven that are based on statistical analyses. The third is the combined model in which intuition is used to determine the basic type of the model and data analysis is used to determine the constants of the model. From another perspective metric models are also classified into process metrics models and product metrics models (Danilecki, et. al, 2011). Process metric models deal with improving the process of software development, while product metric models focus on internal attributes of software product, and relate them to external quality of the product.

The ISO/IEC 9126 standard (ISO/IEC 9126-1, 2001) describes a model for software product quality that dissects the overall notion of quality into six main characteristics: functionality, reliability, usability, efficiency, maintainability and portability. As society becomes more and more dependent on software and demands that new, more capable software be provided on short cycles, the need for reliable software continues to increase (Alagappan et. al, 2009). Therefore recognizing reliability as one of the key factor for software quality this study focuses on the predicting this quality attribute.

1.3 OBJECT-ORIENTED PARADIGM

The emergence and application of object-oriented technology to system development have brought many advantages and benefits as well as new challenging issues. Object-oriented approach offers potential benefits over traditional software development methods and has thereby managed to attract enthusiastic support in industry. Object-oriented technology also provides significant ease in software redevelopment areas (He, et. al, 2012). The main structural mechanisms of this paradigm namely inheritance, encapsulation, information hiding, polymorphism, are the keys to foster reuse and achieve easier reliability.

The object-oriented paradigm has become increasingly popular in recent years. More and more organizations are introducing object-oriented methods and languages into their software development practices. The technology was originated from developments in new computer languages like Smalltalk (Alkadi and Carver, 1998). Today the object-oriented paradigm encompasses a complete view of Software Engineering.

This paradigm suggests that natural objects occurring in a problem should first be identified and then implemented. It also proposes the use of hierarchically related abstract data types to implement a system as collections of highly independent subsystems. These subsystems interact with one another through small, carefully designed interfaces. Each subsystem is unable to assess implementation details of other subsystems. The management of the desired level of visibility in this way is called ‘information hiding’ and is said to yield designs, which have a lower degree of coupling between different subsystems (Arnold, 1993).

Lower coupling is said to translate into greater ability to implement separate systems independently of one another. The stronger the coupling between software artifacts (i) the more difficult it is to understand individual artifacts, and hence to correctly maintain and enhance them; (ii) the larger the sensitivity of unexpected change and defect propagation effects across artifacts; and (iii) consequently, more testing required to achieve satisfactory reliability levels (Bluemke, 2001).

Object-oriented concepts became part of mainstream language technology during 1990s, and there is now a range of languages and products, which offer features to support object orientation. Object-oriented software development is iterative with overlapping phases in the development process. While the basic set of objects, operations, attributes and relationships are identified in analysis phases, the detail of a class's methods, parameters, data declarations and relationships are resolved during design.

Object is the fundamental constituent of object-oriented modeling and design, which combines both data structure and behavior in a single entity (Li, et. al, 2012). Three fundamental characteristics required for an object-oriented approach are encapsulation, polymorphism and inheritance. Polymorphism and Inheritance are two aspects unique to object-oriented approach, while encapsulation is not (Alagappan et. al, 2009).

Encapsulation or Information hiding is a way of designing such that only the subset of the module's properties, is known to users of the module. Encapsulation means, all that is seen of an object is its interface, namely the operations that can be perform on the object (Alkadi and Carver, 1998). Information hiding is a theoretical technique that has undisputedly proved its value in practice. Large programs that use information hiding have been founded to be easier to modify by a factor of 4, than programs that do not (Basili et. al, 1996).

Inheritance is a form of reuse that allows programmer to define objects incrementally by reusing previously defined objects as the basis for new objects. Inheritance decreases complexity by reducing the number of operations and operators, but this abstraction of objects can make maintenance and design difficult. Inheritance favors reuse of standardized software components. Ledbetter and Cox refer this reusable component as to software ICs (Arnold, 1993). As integrated circuits (ICs) have revolutionized electronic hardware design by providing a wide range of subsystems that can be assembled for new applications, reusable software components may also be used in the same manner.

Polymorphism is an important concept that allows the building of a flexible system. This concept of polymorphism allows the developer to specify what shall occur and how. Polymorphism means having the ability to take several forms. For an object-oriented system, polymorphism allows the implementation of given operations to be dependent on the object that contains the operations. Polymorphism also assists the programmer in reducing the complexity (Danilecki, et. al, 2011). Through polymorphic behavior, it is possible to write code that manipulates many different forms of objects in a uniform and consistent manner without regard to their individual differences. The flexibility and generality of polymorphic structure is one of the significant advantages of object-oriented paradigm.

1.4 SOFTWARE RELIABILITY

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment (IEEE 1991). In other words, software reliability corresponds to the successful execution of software operation for a specified period of time under specified conditions. Software quality is the degree to which software possesses desired attributes in meeting laid down specifications. In this scene, software quality encompasses the array of software attributes such as usability, testability, portability, reliability, maintainability and scalability. Software reliability is an important facet of software quality which quantifies the operational profile and life of a software system (Musa, 1975).

The terms error, fault, and failure are often used interchangeably, but these do have different meanings. According to the IEEE (1991) standard, an error is a human action that produces an incorrect result. For example, an incorrect action on the part of a programmer or operator may result coding error, communication error, etc. Fault is an incorrect step, process, or data definition in a computer program. A fault is the materialization of an error in the code. It is a software defect that potentially causes failure when executed (Schick and Wolverton, 1978). Alternatively, a fault is the defect in the program that, when executed under particular conditions, causes a failures. Also, fault is a developer-oriented concept and considered as a property of the program rather than a property of its execution or behavior. Faults may arise across the various stages of software development yielding requirements faults, design faults, and coding faults (Littlewood and Verrall, 1973; Pandey and Goyal, 2013).

A failure is a departure of system behavior from user expectation during execution. A software failure is said to occur when the user perceives that it deviates from its expected behavior. Then, the software reliability is determined as the probability of software failure for a specified period of time in a specified environment. Hence, the software failure and reliability are more close to end user than developer (Chatterjee et. al, 1997; Pandey and Goyal, 2013). Potential failures found by the users or developers are traced back to find the root causes such as requirement faults or design faults or coding faults. It is the developer who identifies the root cause of these failures by tracing back to program design or code and find associated faults and errors. The reliability is usually influenced by finding and fixing these faults that causes failures. Thus, software error, fault, failure, and reliability have different views of software developers, testers and users point of view (Ritika, et. al, 2012).

An error results in fault which is responsible for causing failure upon execution. The reliability of a system can be determined by finding the probability of these failures. In brief, error causes faults and a fault causes failures which are responsible of affecting software reliability. In other words, a fault is created when human makes an error. Errors may be in the requirement document, in the design document, in the code, or in the testing process causing faults in the developed software. These faults cause failures when got executed (Pressman, 2005). Failures may be categorized as transient, permanent, recoverable, unrecoverable, non-corrupting, and corrupting based on their effect. Modeling and analysis of these software failures are important in software reliability prediction.

1.5 FUZZY LOGIC

An overall fuzzy logic system for reliability modeling is shown in Figure 1.1. There are four main components of a fuzzy logic system, namely fuzzification process (input membership function), fuzzy rule base, fuzzy inference process and defuzzification (output membership function). The fuzzification is the process of mapping crisp value to the fuzzy value. Depending on the available data and associated uncertainty, the input and output variables are fuzzified in terms of linguistic variables such as low (L), medium (M) and high (H). The basic unit of any fuzzy system is the fuzzy rule base. The failure analysis and human knowledge expertise together with historical data constitute the fuzzy rule base. All other components of the fuzzy system are used to implement these rules in a reasonable and efficient way (Yadav et. al, 2003).

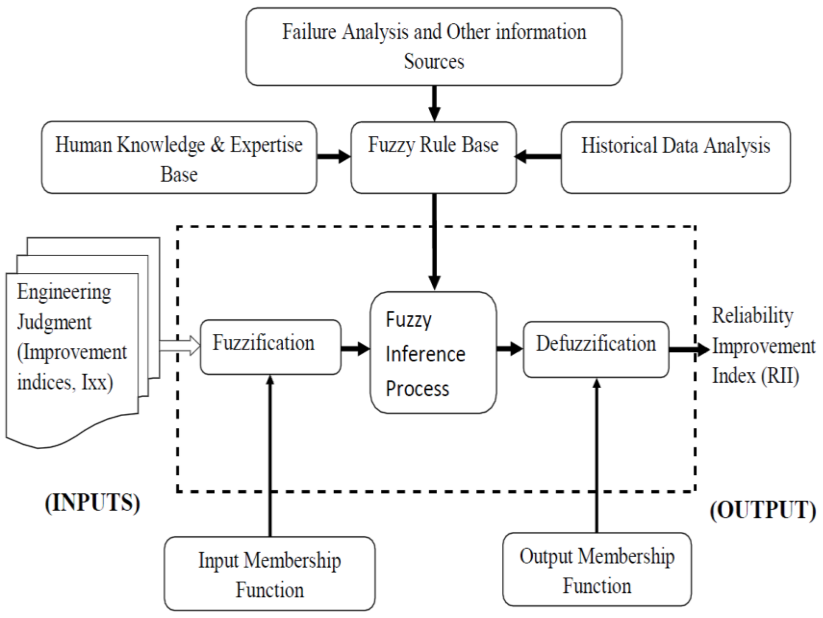
Failure Analysis and Other information Sources
Human Knowledge & Expertise Base
Fuzzy Rule Base
Historical Data Analysis
Engineering Judgment (Improvement indices, Ixx)
Fuzzification
Fuzzy Inference Process
Defuzzification
Reliability Improvement Index (RII)
(INPUTS)
(OUTPUT)
Input Membership Function
Output Membership Function

Figure 1.1 Overview of Fuzzy Logic System for Reliability Modeling

The fuzzy inference process combines the rules in the fuzzy rule base and then provides a mapping from fuzzy input (input membership function) to fuzzy output (output membership function). After fuzzification these data and information are used for defuzzification to get the reliability index. Defuzzification is the mapping of a fuzzy quantity to a precise value, like the fuzzification is the conversion of a precise quantity to a fuzzy quantity (Pandey and Goyal, 2013).

1.5.1 Fuzzy Logic Vs Formal Methods

Formal methods are widely recognized as a powerful engineering method for the specification, simulation, development, and verification of interactive systems (Manfred and Ketil 2001). They follow the principle of “correctness by construction” and are therefore well suited for security-critical systems (Anthony and Roderick, 2002). However, most formal methods rely on a two-valued logic, and are therefore limited to the axioms of that logic: a specification is valid or invalid, component behavior is realizable or not, safety properties hold or are violated, systems are available or unavailable. Although the promises of formal methods are well known (Luqi and Goguen, 1997), there are many limitations preventing the usage in industrial software development. The following limitations are generally identified in literature (Pavol et. al, 2014; Ian, 2006) as the main blockers: 1) Limited scope: Formal methods are not well suited to specifying user and environment interfaces and interactions; 2) Limited scalability: As systems increase in size, the time and effort required to develop a formal specification grows disproportionately; 3) Limited expressiveness: standard formal methods are not capable to quantify values between the “absolute truth” and the “absolute false”.

Through the longtime experience obtained within the research projects SPES (Klaus, 2012) and E-Energy1, Vasileios Koutsoumpas (2015) empirically confirmed the presence and challenges of the above stated limitations for the avionic, automotive, and smart grid domain. In recent years, there is a number of research attempts (Chris and Paul, 2000; Matthews, 2002; Pavol, 2010; Thomas and Jan 2014), which point out the need for emerging ideas and concepts to overcome the limitations for formal Methods. Especially when the problem domain entails uncertainty, impreciseness, and vagueness, the appliance of formal methods becomes a challenging task. In order to overcome the limitations resulting from the strict modus operandi of formal methods, In a study (Vasileios, 2015) author has relax the boolean notion of formal specifications by using fuzzy logic. With respect to formal specification based on fuzzy logic, Chris and Paul (2000) suggest fuzzy set theory as a possible representation scheme to deal with uncertainty.

In (Lopez, 2008) author showed how fuzzy logic techniques can be used to write software specifications in a fuzzy framework, as a previous step in the process of analysis and design of new software. The study has used an example of a multiprocessor system to show how information closer to the natural language can be managed in terms of fuzzy sets and fuzzy inference rules. In a study (Victoria, 2011) author shows a new proposal to formalize the software development life cycle and the importance of using not only the classical logics (propositional and first order logic, that is used in formal methods), but also fuzzy logic in order to formalize the certain and uncertain information involved in natural language (Rizvi and Khan, 2013).

The alternative that was recommended by above researchers is the use of fuzzy logic techniques to tackle with the limitations of formal Methods. The fuzzy set theory has emerged as an alternative to capture the vagueness, uncertainty and imprecision present in the natural language. Therefore, in early stages of product development, where the data is inadequate or is present in form of ‘knowledge’, use of fuzzy logic would be more appropriate than formal methods. Any model based on Fuzzy Logic help in the prediction of software residual defects.

In order to deliver reliable software, Fuzzy Logic techniques are playing a critical role. These techniques are emerging as robust optimization techniques that can solve highly complex, nonlinear, correlated and discontinuous problems. As the most of the early stage software metrics are not very comprehensible and involve high and complex dependency among them. That’s why fuzzy logic inference systems have found usefulness in capturing and processing subjective information in terms of software metrics in the early phase of software development. Therefore, fuzzy logic is considered to be an appropriate tool in these situations (Rizvi, et al, 2016d).

1.5.2 Fuzzy Logic Vs Probability

Suppose you are a basketball recruiter and are looking for a ‘‘very tall’’ player for the center position on a men’s team. One of your information sources tells you that a hot prospect in a city has a 95% chance of being over 7 feet tall. Another of your sources tells you that a good player in another city has a high membership in the set of ‘‘very tall’’ people. The problem with the information from the first source is that it is a probabilistic quantity. There is a 5% chance that the player is not over 7 feet tall and could, conceivably, be someone of extremely short stature. The second source of information would, in this case, contain a different kind of uncertainty for the recruiter; it is a fuzziness due to the linguistic qualifier ‘‘very tall’’ because if the player turned out to be less than 7 feet tall there is still a high likelihood that he would be quite tall (Ross, 2010).

Another example involves a personal choice. Suppose you are seated at a table on which rest two glasses of liquid. The liquid in the first glass is described to you as having a 95% chance of being healthful and good. The liquid in the second glass is described as having a 0.95 membership in the class of ‘‘healthful and good’’ liquids. Which glass would you select, keeping in mind that the first glass has a 5% chance of being filled with non healthful liquids, including poisons? What philosophical distinction can be made regarding these two forms of information? Suppose we are allowed to measure the basketball players’ heights and test the liquids in the glasses. The prior probability of 0.95 in each case becomes a posterior probability of 1.0 or 0; that is, either the player is or is not over 7 feet tall and the liquid is either benign or not. However, the membership value of 0.95, which measures the extent to which the player’s height is over 7 feet, or the drinkability of the liquid is ‘‘healthful and good,’’ remains 0.95 after measuring or testing. These two examples illustrate very clearly the difference in the information content between chance and fuzziness (Ross, 2010).

1.6 OBJECTIVES OF THE STUDY

Most of the reliability models currently available can be applied only after a product is complete, or nearly complete. This provides information too late to help in improving internal product characteristics prior to the completion of the software product. Therefore, there is a need for metrics and models that can be applied in the early stages (requirements and design) of development to ensure that the software design have favorable internal properties that will lead to the development of quality end product. This early feedback would provide an opportunity to make appropriate corrections, remove irregularities and reduce complexity early in the development life cycle.

In view of aforementioned need as well as advantage of early reliability prediction, the research has identified following objectives:

a) To evaluate the feasibility of predicting software reliability through Fuzzy inference System on the basis of requirements and design stage measures.

b) To identify the product-oriented measures at the requirement stage those could impact the reliability of the final software to be delivered.

c) To identify object oriented design construct as well as metrics those are contributing towards predicting class diagram's reliability.

d) To develop a comprehensive framework that can provide a roadmap for the implementation of early reliability modeling as specified in the first point (a).

e) To develop and validate Early Stage Reliability Prediction Model (ESRPM) that predicts the reliability of the developing software at its design stage (i.e. before the coding starts) through FIS.

1.7 RESEARCH PROBLEM

Because of the high user expectation and heavy investment during the development, reliability has become an important concern for software developers as well as customers. One of the observations that cannot be overlooked is the need of timely identification and subsequent fixation of residual defects so that reliable software could be delivered in time. The best time to detect and arrest faults is the requirements and design stages. To accomplish this task researchers are bound to use quality measures based on these stages. But usually most of metric values in early stages are subjective as their sources are the opinions of domain experts.

Therefore, to deal with such intrinsic subjectivity and vagueness, fuzzy techniques have come up as a dependable tool in capturing and processing these early stage metric values. There are just a few attempts where fuzzy techniques were used to quantify the reliability. But the key concern is the time and the stage of SDLC. These models are helping developers either by the end of coding phase or in the testing stage. These feedbacks make it too late to improve the existing product towards a more reliable one. Therefore there appears a need to make an early effort to further improve the modeling of Software Reliability using the potential of Fuzzy Logic.

Keeping in view of the objectives illustrated, the following research questions have been set forth. The researcher will try to find out solutions for the questions posed hereunder.

a) Is there any reliability prediction model available for object- oriented software at the design stage?

b) Is there any fuzzy based reliability prediction model available that predicts the reliability of the developing software upto its design stage, using the requirements and Object-Oriented design measures (before the coding starts)?

c) Is it feasible to develop an early stage reliability model as specified in point (b)?

d) Are there any product based measures at the requirements stage those may impact on the reliability of the developing software?

e) Is it justified to engage the fuzzy technique in order to develop a reliability model as specified in point (b)?

f) Are there quantifiable features that can be extracted from information available in early stages that can be used to help predict software reliability?

g) Is there any framework or roadmap exists that may guide the researchers or industry professionals in developing a model as specified in point (b)?

The researcher has made an effort to answer for the above cited questions in Chapter 7, by formulating the research problem as follows:

"Improving the Modeling and Implementation of Software Reliability using Fuzzy Logic Techniques"

1.8 METHODOLOGY

Two phases of SDLC i.e. requirements and design are focused in developing the Early Stage Reliability Prediction Model (ESRPM). The development process of the model is comprised of following steps. The first step starts with conceptualization. Critical review of various reliability prediction studies has been put forth as second step followed by the steps termed as developing the framework, Implementing the framework, theoretical and empirical validation, predictive accuracy of the model and comparison of newly developed model with earlier relevant studies.

1.8.1 Conceptualization

The development process starts with the conceptualization. That is the initial brainstorming activity envisaged and undertaken to understand the problem. Importance of this step lies in the fact that it serves as a basis for the subsequent steps.

1.8.2 Literature Review

A critical review of the literature will provide information regarding, what has been done in the same area, which will lead to significant investigation. Detailed and careful reading of the reviews of expert researchers would also promote greater understanding of the topic, procedures, methods and enable to frame useful hypothesis.

1.8.3 Developing the Framework

Developing the framework (Fuzzy Logic based Software Reliability Quantification Framework (^{FL}SRQF).) is an important activity prior to the model building, because it guides the entire development of the proposed model. The framework is comprised of eight phases.

1.8.4 Implementing the Proposed Framework

This is a very crucial phase of the methodology, as it implements all the eight phases of the Fuzzy Logic based Software Reliability Quantification Framework (^{FL}SRQF).

1.8.5 Theoretical and Empirical Validation

Theoretical validation of the developed model provides the supporting evidence as to whether a model actually measures what it purports to measure. In empirical validation proper statistical analysis will be used to ensure the effectiveness of the developed model.

1.8.6 Predictive Accuracy

An important aspect of any model is its predictive accuracy. It ensures that the developed model is predicting the values of dependent variable accurately. The thesis will also measure the predictive accuracy of the developed model (ESRPM).

1.8.7 Comparison of Developed Model with Existing Reliability Prediction Studies

Whenever a new quality model or a new metric suite is being developed, it is required to compare that model or metric with the existing models or metrics of the same nature.

1.9 SIGNIFICANCE OF THE CONTRIBUTION

As it is known that the requirements as well as design phase plays a very crucial role in the development life cycle of quality software. The role of class diagrams in the design of object-oriented software is very critical. Therefore quality of class diagrams has a significant influence on the quality of the object-oriented software that will be finally delivered. This study has been concentrating on the key factors of requirements and design phase that are impacting the reliability of the developing software. Following are the some points highlighting the significance of this study:

a) One of the major significant contribution of this research is the Fuzzy Logic based Software Reliability Quantification Framework (^{FL}SRQF). The framework is quite prescriptive in nature, and will definitely facilitate industry professionals and researchers to predict reliability in the early stage, and subsequently decrease the probability of software's unreliability.

b) Consideration of the requirements phase along with the design provides this research an edge over other studies those are based on only design phase, because ignoring requirements deficiencies and only concentrating on making design constructs superior will not seems good enough.

c) The suitability of various requirement and design measures as a contributor for the software reliability has been identified.

d) The reliability model developed as per the guidelines of the proposed framework may help software professionals to take appropriate corrective measures right from its requirements phase, to deliver software with an improved reliability level, close to the user's expectation.

e) Based on the analysis of quantified values the research assists developers by providing them an opportunity, to improve requirements and design related internal characteristics ahead of writing the final code.

f) The proposed model may help designers as well as developers to predict the reliability of the developing software upto its design stage, early in the development life cycle.

g) Based on the predicted reliability of developing software upto its design stage, the developers may predict the reliability of the final software to be delivered.

h) In order to overcome the limitations of subjective values of requirements metrics, the research has utilized the strength of fuzzy inference process in its quantification phase.

i) The 'Assessment and Amendment' phase of the framework further strengthens it practicality as well as viability by keeping the doors of improvement open for any of the earlier phases.

j) In most of the cases, developed models only provide quantitative values but neither provides suggestions on how to make improvement, nor the precautions on how to avoid abnormalities. Therefore, to fill this gap this research recommends to provide needed suggestive measures based on the results and contextual interpretations.

k) Apart from the above, reassessment of previously developed or underdevelopment reliability quantification models could be done as per the guidance of the proposed framework.

l) Beside this, as far as further research is concern, the model may open fresh avenues for the researchers, doing research on reliability estimation as well as reliability prediction of object-oriented software.

1.10 LIMITATIONS

Every effort has some limitations. Nobody can perform a perfect work. This section of the dissertation points out some of the limitations of the study:

a) Our approach assumes that the SRSs are written in plain English text, a primary form to state requirements. Our approach may not be applicable for SRSs specified in some formal languages.

b) The research focuses only on requirements and design phase of object-oriented software.

c) The model proposed in this study quantifies the reliability of object-oriented software only.

d) The research considered only four object-oriented design constructs (i.e. Encapsulation, Coupling, Inheritance and Cohesion) to predict the reliability.

e) The model is validated through a set of twenty projects of varying reliability, similar experiments with more quality projects from industry may be carried out to get more generalize results.

f) Reliability level of the industry projects (used for validation of the model) is subjective and based on the feedback from the clients, number of failure reports, types of maintenance, etc.

Although the reliability model (ESRPM) can be applied in a wide range of object-oriented software, it has not been claimed that this is the only model for predicting reliability, nor it is the best. More precise and generalize model can be developed following the process discussed in this dissertation.

1.11 THESIS OUTLINE

The first chapter presents an introduction of the study. It highlights the background as well as need of this research. It also provides conceptual information about Software Quality, Software Reliability and Object-Oriented Paradigm. The chapter also highlights the objectives of the study, methodology used, significance as well as limitations of this study.

The second chapter starts with the description about the current situation, highlights the virtues of early reliability prediction. Then it presents a comprehensive state-of-the-art on software reliability prediction as well as estimation, followed by a summary of critical findings.

The third chapter proposes a structured framework that may overcome the inadequacies of earlier studies and quantifies the reliability, on the basis of the requirement and design phase measures, before the coding starts. Salient characteristics of the framework have also listed at the end of the chapter.

The fourth chapter implements all the eight phases of the framework proposed in the third chapter. The developed fuzzy based reliability model has been validated theoretical as well as statistically in chapter fifth. Predictive Accuracy of the model is also presented in Chapter 5.

The sixth chapter presents a detailed comparison of the Reliability Model with the existing reliability Models. The conclusion, major findings and the recommendations for future enhancements are summed up in last and Seventh chapter.