CHAPTER 2

RELIABILITY ESTIMATION AND PREDICTION: A REVISIT

2.1 BACKGROUND

Research in a particular field requires an elaborate review and study of literature related to that subject. A critical review of the literature provides information regarding, what has been done in the same area, leading to a significant investigation. Detailed and careful reviews of the experts and researchers also promote greater understanding of the chosen topic, procedures, methods and algorithms and enable to frame useful hypothesis (Duraisamy, 2008).

As revealed in the literature (Bhatnagar and Kakkar, 2015; Catal, 2011; Dhiauddin, et. al, 2012; Fazal-e-Amin, et. al, 2012; Fenton et al., 2007; Li, 2002), the cost of a software application in the past decades was sweat, blood, tears, and endless debugging sessions. This is because the demand for complex software systems has increased more rapidly than the ability to design, implement, test, and maintain them. Besides, the ever increasing complexity of software has impaired our ability to understand how faults are born, manifest, propagate, and eventually lead to failures of the software. Many reported system outages or machine crashes were traced down to

computer software failures, such as the London Stock Exchange crash in 2008, the Air-Traffic Controller incident at Los Angeles Airport in 2004 (Kong, 2009). As literature has reported various software problems, the reliability of software systems has become a major concern for our modern society.

Review of the literature has exposed number of unfortunate events that occurred because of lack of reliability in the corresponding software applications, especially in the defense and health domain. In most of the cases a significant number of human had lost their lives. One incident was reported in 1993 by South West Thames regional health authority on the computer aided dispatch system that was broken after its installation, consequently London ambulance service was unable to handle more than 5000 daily request for carrying patients in emergency situations (Li, M. and Smidts, 2003). Due to incompatible software response to the pilots, airline industry had faced a number of airline crashes (Ashish, et. al, 2012). During the Iraq war, an example of software unreliability came into the light when the patriot missile fails to intercept a scud missile (Lyu, 1996). After noticing these events it could be easily concluded that reliability is one of key quality attribute of any software application. Actually, reliability measurement in software is still in its infancy (Lyu, 1996). Even though researchers agree that development process, faults and failures found are all factors related to software reliability, no good quantitative methods have developed to represent software reliability without excessive limitations.

This chapter starts with the background of current scenario. Brief taxonomy of software reliability measurement models has been put forth. Criticality of the early reliability measurement has been discussed, followed by a comprehensive literature

review of various reliability estimation or prediction studies. The chapter presents a survey of Software Reliability prediction along two perspectives; Software Metrics and Fuzzy Logic. Finally the summarized form of critical finding noticed during the review concludes the chapter.

2.2 BRIEF TAXONOMY OF SOFTWARE RELIABILITY MEASUREMENT MODELS

The current practices of software reliability include two categories of activity: reliability prediction and reliability estimation (Lyu, 1996), shown in figure 2.1:

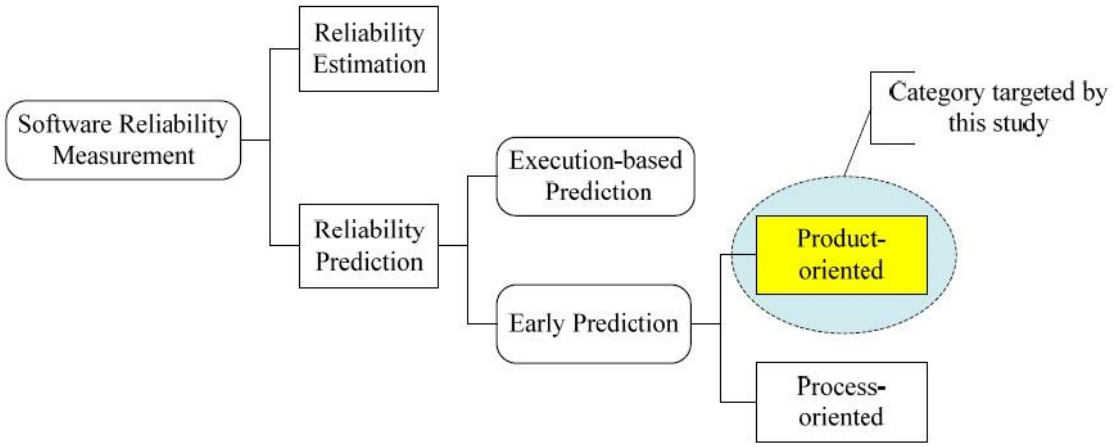

Figure 2.1 Brief Taxonomy of Software Reliability Models

Reliability Estimation: This job determines the software reliability through statistical inference techniques. Techniques are applied to failure data that is obtained while the software is being tested or while, it is in operation. This is a measure regarding the achieved reliability from the past until the current point. Its key objective is to assess the current reliability, and to know whether a reliability model is a good fit in retrospect.

Reliability Prediction: This task finds future reliability of the software, based on available software metrics and measures. Depending on the phase of the software development life cycle, prediction uses different techniques: When failure data are available, the estimation can be based on parameters and verify software reliability models, which can perform future reliability prediction. When failure data are not available, then those metrics are used, that are based on these early stage of software development life cycle like requirements and design stage and the characteristics of the resulting product like software requirements specification document or design documents to determine reliability of the software upon testing or delivery. This is usually called "early prediction" Figure 2.2 (Kong, 2009)

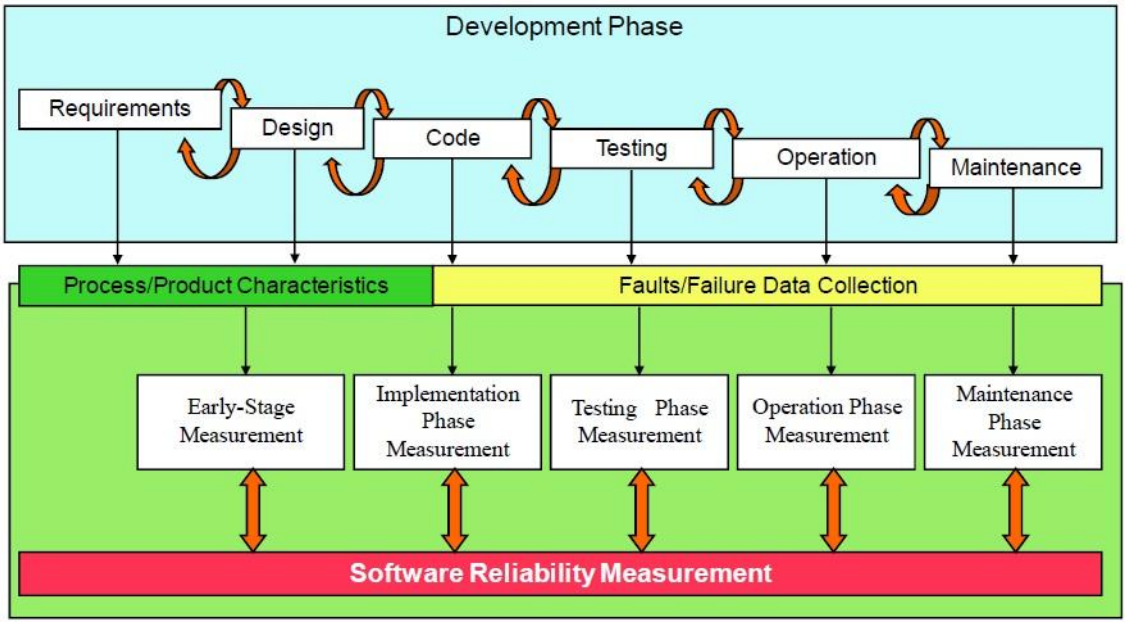

Figure 2.2 Software Reliability Measurement vs. Development Process

Fenton and Pfleeger, (1998) classified software metrics into three main types: product metrics, process metrics, and project metrics: Product metrics describe characteristics of the software development life cycle processes outputs such as requirements specifications documents, design diagrams, source code, and

executable programs. Examples of classical product oriented metrics are McCabe's Cyclomatic Complexity, Line of Code (LOC), and Mean Time To Failure (MTTF) (Neumann, 1988; Smidts and Li, 2000). Process metrics focuses on the attributes of the software development process like the methodology being followed and other factors related to the development environment. Research has highlighted that a relationship exists between the development process and the ability to complete projects on fix schedule and as per the desired quality objectives (Boehm, 2000).

Higher reliability level can be achieved by using better and matured development process, process to handle risks, processes taking care of configuration management, etc. Project metrics are the metrics that are based on the available resources, like the number of developers and their skills. However these metric are rarely used in software reliability measurement.

2.3 WHY EARLY RELIABILITY MEASUREMENT IS NECESSARY

As reliability has become a critical factor in software systems, its prediction is of great importance. An accurate estimate of reliability can be obtained during the last stages of development lifecycle. However, quantifying software reliability in its early stage is one of the key area of concern, as described in the following:

1. The majority of software projects fail to achieve schedule and budget goals.
2. The majority of faults have their root cause in poorly defined requirements.
3. The cost of fixing a software fault is lowest in the requirements phase.

2.3.1 Majority of Software Projects Fail to Achieve Schedule and Budget Goals

Studies available in the literature had described the industry reality for decades: a significant number of software projects failed to achieve the decided schedule as well as the allocated budget. Figure 2.3 taken from (Kong, 2009) shows some significant facts. It could be easily inferred from the following figure that only two percent of the developed software has reached to their operational life as delivered by the developing organization, while majority, of the software (75%) was either never used or was cancelled before delivery.

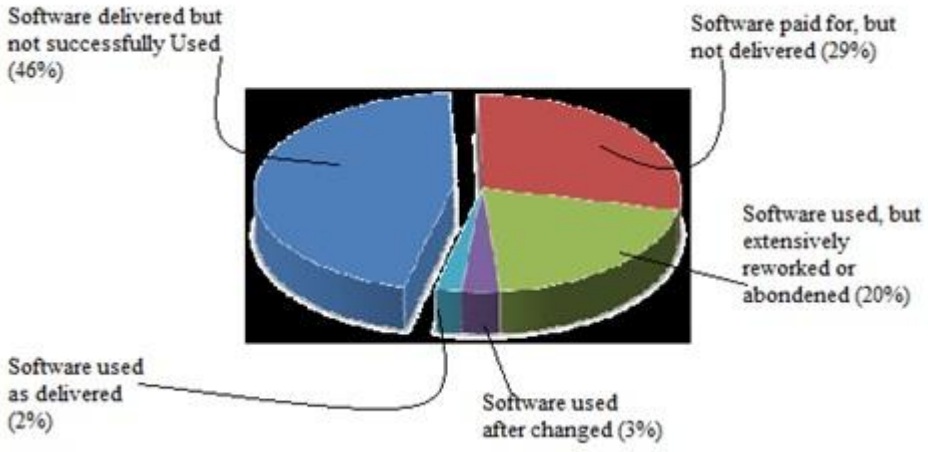

Figure 2.3: Outcomes of Department of Defense Software Spending

A similar work conducted by the Standish Group (1995) on non Department of Defence projects produced similar results. Out of over 8,000 projects handled by 350 companies, 28% of projects has failed, 46% are challenged, and only 26 percent of the projects were qualifies as successful. Poor software quality is a prime factor behind various failures, and results in major rework regarding application scope, design and code (Standish, 1995). Such rework extends release cycles and consumes

significant additional budget. Apart from the time and money that has been further devoted for rework, the more important issue is the impact on the business reputation and its market credibility that compromised because of this rework. Therefore in order to control software failures, it is needed to better understand the quality of products being developed for today's global economy.

2.3.2 Requirements are the Root of Many Problems

There are sufficient studies in the literature that early stages of the software development are prone to defects. The major risks for the software's success are confusion and misunderstanding about the requirements. (Easterbrook and Paul, 1998).

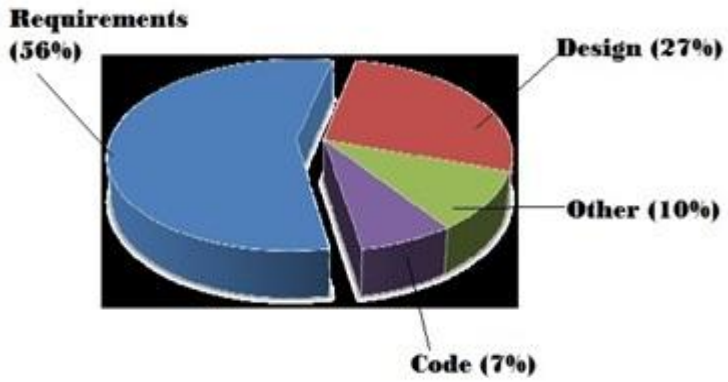

Figure 2.4: Distribution of Faults in Software Projects

In a study of a US Air Force project by Sheldon (1992), defects were classified by their origins. It had found that software requirements comprised 41% of the total defects, while design faults are only 28% of the total faults. Other studies also support this result as well. For example, a study conducted by James Martin (1986) had reported that over half of all project defects have their roots in the requirements stage as indicated in Figure 2.4 (adapted from (Martin, 1986)). Further, the study stated that

approximately fifty percent of requirements defects originated from the poorly written, ambiguous and incorrect requirements. The rest fifty percent faults could be attributed to the requirements specifications that are incomplete or those were just omitted.

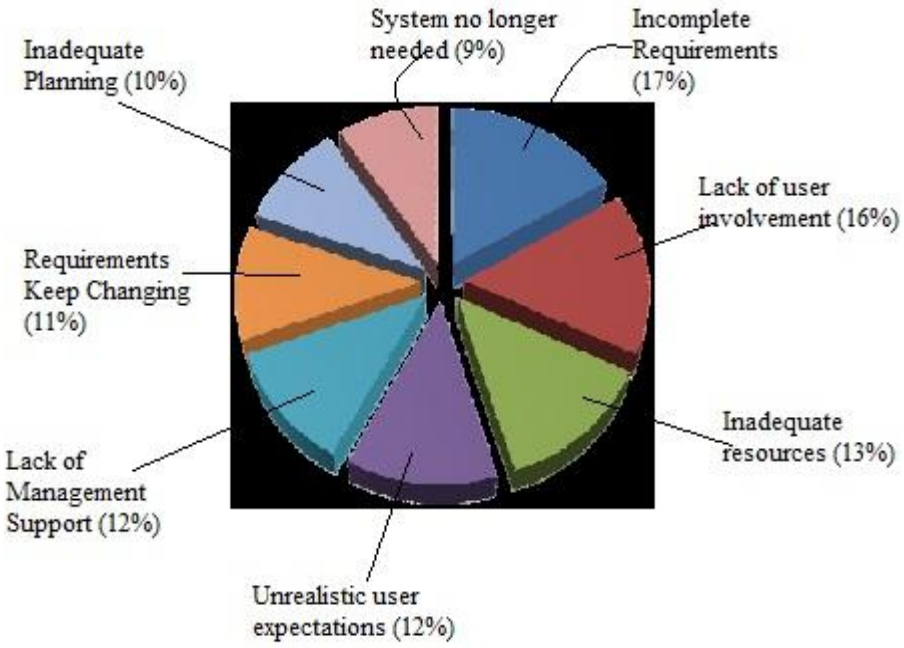

Figure 2.5 Distribution of Failure Causes of 8000+ projects

Other statistics exhibited comparable problems:

- 70-85 percent of application rework was related to defects in requirements (Martin, 1986)
- 44% of projects were cancelled because of problems with requirements (Martin, 1986).
- 54% of initial project requirements were actually realized (Standish, 1995; Kong, 2009)
- 45% of realized requirements ended up actually being used (Standish, 1995)

A survey of Standish Group (1995) also found that of the eight main causes for failures of software projects, out of these eight reasons, five were related to poor requirements, as presented in Figure 2.5 (adapted from (Standish, 1995)). The issues responsible for these poor requirements include incomplete requirements, absence of user involvement, impracticable user expectations and frequencies of change requests. Various studies also highlighted that process of gathering the requirements is also one of the cause for them. Therefore getting the right requirements is probably the most important thing that should be done appropriately to achieve customer satisfaction.

2.3.3 Faults Cost Less when Detected and Fixed in Early Stages of Development

The importance of requirements is further emphasized that bulk of the effort (82%) is attributed to fixing requirement defects (Leffingwell and Widrig, 2000). It is an accepted fact by the most practitioners that cost of fixing a defect is lowest at the requirements stage. As the software development progresses into subsequent stages, the fixation cost of fault increases dramatically, since this fixation affects other software deliverables like design document, source code or test cases. The earlier a fault is detected, the less damage it can do to the software, because only very few deliverables need corrections. According to the industrial data, the cost of detecting and fixing a defect that is introduced during the requirements and design stage of the software development life cycle increases exponentially as the development progresses through the later stages (Graham et. al, 1993). This fact is pictorially represented in the following figure 2.6. In another study, McConnell (1996) concludes that "a requirements fault that is left undetected until construction or maintenance will cost 50 to 200 times as much to fix as it would have cost to fix at requirements time."

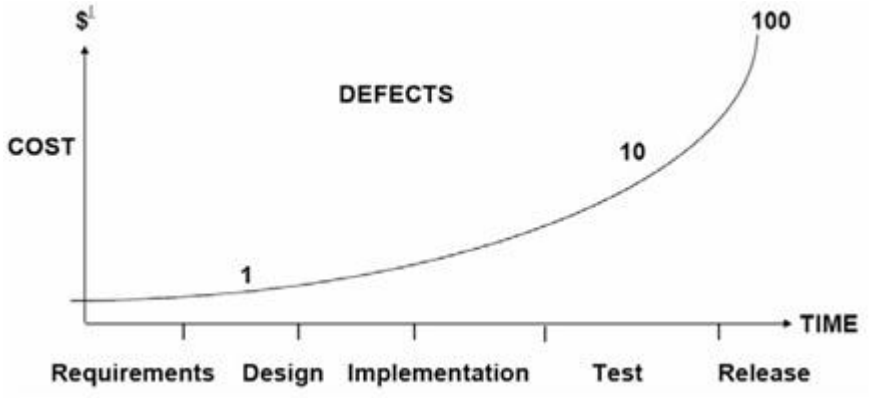

Figure 2.6 Industry Standard Cost Ratio to Fix a Defect

Other studies, furthermore, show that requirements faults are between 10 and 100 times more expensive to fix during later stages of the development life cycle than during the early stage itself. The reason for this significant difference is the reason that most of these defects are not discovered until well after they have been made. This delay in fault discovery means that the cost to repair includes both the cost to correct the offending fault and the cost to correct subsequent investments in the faults which were made in later phases (Kong, 2009).

These investments include the cost for redesign and replacement of code, cost for rewriting and preparing the changed and updated documentation. In fact, the major issue is scrap and rework. If a defect was introduced during the coding phase then the coder just fix the error and re-compile the code. But, if a fault has its origin in the requirements and not been detected until the testing stage then the rework has increased significantly. Because, developer re-do the requirements, then correct the design, re-do the coding, revise the test cases as well as the related documentation. It is all this "re-do" work that sends projects over budget and over schedule.

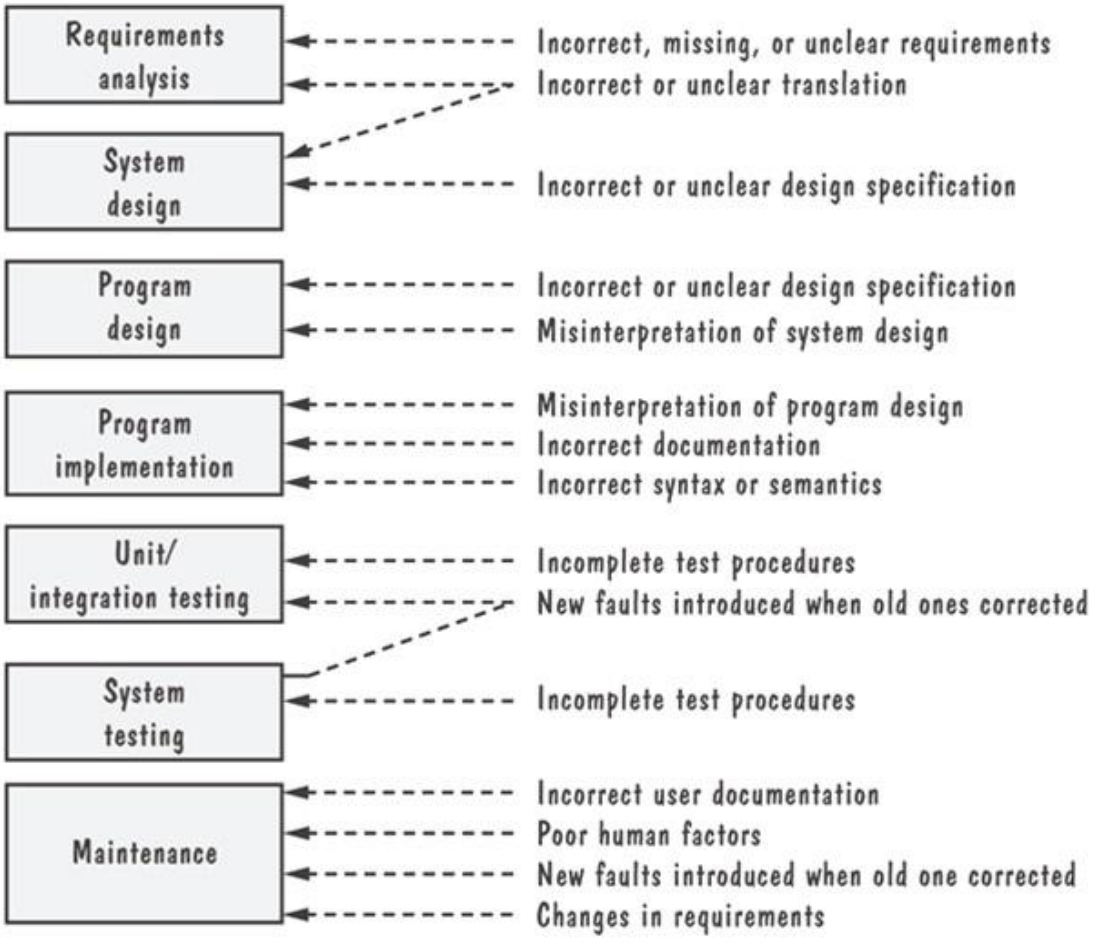

Figure 2.7 Cumalative Effect of Faults

This claim is supported by many studies. For instance, a study reported that approximately 40% of the entire software budget was spent in rework or revision because of faults detected late in the development life cycle. Another study (Davis and Leffingwell, 1995) indicates that in the current scenario majority of companies spent between 30-40% of total project costs on rework activities. Subsequently finding and fixing faults consumes 70% - 85% of total costs. Faults are introduced in various stages of the development process, as shown in Figure 2.7 (taken from (Pfleeger and Atlee, 2006)). This figure shows that faults which originate in early stages can have a lasting

influence on the quality of software: they are the earliest to invade the system and the last to leave, if not fixed. This is referred as fault cumulative effect (Pfleeger and Atlee, 2006), which highlights why requirements stage defects, in comparison to defects introduced into the design and coding phases are generally more expensive to be defected and fixed.

The role of software has shifted from simply generating financial or other mathematical data to monitoring and controlling equipment which directly affects human life and safety. Software's increasing role creates both requirements for being able to trust it more than before, and for more people to know how much they can trust their software products (Kong, 2009). As a result, methods used to achieve, predict, and assess the safety and reliability of software is strongly needed in academia, industry, and government. This is also true since many legal issues related to software liability are evolving (Kong et. al, 2007).

Different parts of the software-related industry and society face different challenges. For developers, designers and managers involved in software development it become necessary that they identify as well as recognize early indicators for the development of quality software. These indicators will provide an opportunity to take in time corrective measures to reduce cost and prevent disasters. Similarly for regulators and policy makers involved in the certification of software systems, practical methods and tools are also needed for quantitatively assessment of software products (Tyagi and Sharma, 2012). Clearly, software engineering suffers from problematic requirements specifications. Matured, well-defined, and quantitative

assessment methods for the reliability of the software products are not generally applicable until later life cycle phases. Most prediction methods prescribed for early stages remain qualitative and depend heavily on expert opinions and their subjective judgments. Therefore, the need to develop better software requirements engineering techniques is urgent (Kong, 2009).

2.4 VIRTUES OF EARLY SOFTWARE RELIABILITY MEASUREMENT

First, the advantage for early software reliability measurement is simple economics. Defects introduced during requirements are the major source of project failures and the most costly to be fixed. Because of that, not only detection of requirement defects, but also their removal at the earliest during the development life cycle will significantly improve the quality of the developing software to be delivered in future. With the cost of some software exceeding tens or even hundreds of millions of dollars and with development time of more than 12 to 18 months, early reliability measurement can significantly contribute to the success (Fazal-e-Amin, et. al, 2011).

Secondly, early software reliability measurement provides a solid groundwork and at the same time 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. If software reliability assessment is done early then it is possible to determine what corrections or improvement can be made to the software methods, techniques, or organizational structure.

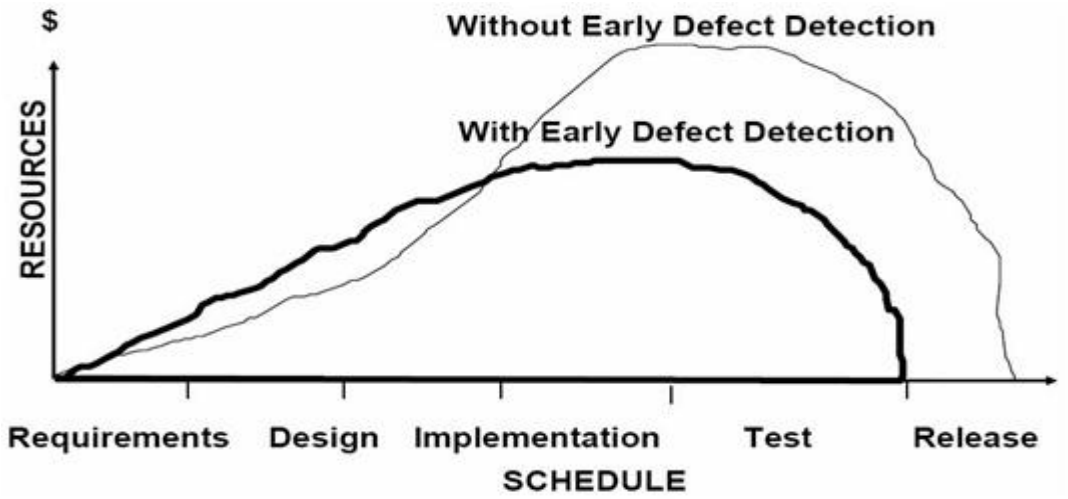

Figure 2.8 Development Schedule with/without Early Fault Detection

Thirdly, with recent strong emphasis on the development speed of software, the early feedback can have the greatest impact on schedules of software projects. It was observed that early defect detection could significantly cut down the development schedule, as shown in Figure 2.8 (taken from (Fagan, 1986). This is because the future rework is reduced if requirements defects are detected and removed during early stages of development (Fagan, 1986; Smidts et. al, 1988).

2.5 STATE-OF-THE-ART ON RELIABILITY MEASUREMENT

During the last two decades significant number of software reliability prediction and defect prediction models has been proposed, where the researchers have used a variety of techniques to achieve their objectives (Arikan, 2012; Ashish, et. al, 2014a; Dambros, et. al, 2012; Hai, et. al, 2013; Palviainen, et. al, 2011; Pandey, et. al, 2012; Si, et. al, 2011; Tyagi and Sharma, 2014b; Ying, et. al, 2014). Early software reliability prediction has attracted significant interest from software practitioners and researchers since the early 1990's. However, quantifying software reliability in an early

stage has been a difficult research subject that many researchers have attempted to solve with limited success (Rizvi, et al, 2015; Tyagi and Sharma, 2014c). Traditional methods for predicting the software reliability such as reliability growth models, base their estimates on the number of observing defects as well as fixing faults during validation testing, where operational patterns represent the environment as the product will face during its operational life in the actual field use. Unfortunately, during early stages of software development failure data is absent to predict or estimate the reliability of the developing software. Therefore, literature has witnessed a verity of approaches and models, but only a few can be applied in early development stages, e.g. requirements and design stage, before the coding starts. This is because only those methods or models that provide a reasonable estimation without the need of any actual failure data are applicable in early development stages.

The pioneering early-stage reliability measurement models proposed in the early 1990's include: Gaffney and Davis' (1990) phase-based model, Agresti and Evanco's (1992) Ada software defects model, and the US Air Force's Rome Lab model (Rome Laboratory, 1992). The basic thinking of these early-stage models is to obtain as much information as possible. Kong, (2009) has referred this approach of early prediction as the "white box" approach, which requires detailed information usually not available in most cases. There are parameters in the estimation and prediction model that have tradeoff capability (maximum/minimum predicted values). The developer can determine where some corrections or improvements can be made in development process to achieve better and improved estimates for fault-density. However, this tradeoff is valuable only if the analyst has knowledge of the software development process.

Smidts et. al, (1997), had used Bayesian networks to make out how software metrics are related with defect proneness? The study had used metrics data from the promise repository. Beside the software metrics available in promise repository, authors defined two more metrics NOD (based on the number of developers) and LOCQ (related to the quality of source code). On the basis of the results from the experiments, the study had inferred that, on defect proneness, the three metrics CBO, WMC, and LCOM were not as effective as RFC, LOC, and LOCQ. Besides that the study had also highlighted that the two metrics NOC and DIT had very restricted effect and are undependable.

Okutan and Yildiz, (2014) proposed an early defect prediction model with Bayesian Nets, which may predict defects that may pop up either during testing or when the software would be used during its routine operations. The study applied the model by analyzing numerous evaluation measures on a dataset obtained through a questionnaire distributed to thirty one, consumer electronics project managers.

Zheng, and Lyu, (2010) suggested how metric's subjective assessment can be done with the help of a questionnaire. The study also mentioned that few of the project factors, like size of the software with some metrics may predict the software residual defects.

Tripathi and Mall (2005) proposed a model based on Reliability Block Diagram (RBD) for representing real-world problems and an algorithm for analysis of

these models in the early phases of software development. The authors had also reiterated that most of the fault prediction models rely on the software size and complexity metrics for estimating the number of defects, and used for system design assessment.

By assuming the same failure rate between two similar software projects, Hu et. al, (2006) suggested to "reuse" failure data from previous releases. The authors has observed better prediction level in the early phases of testing compared with the original ANN model without failure data reuse (Kong, 2009).

Brosch et. al, (2010) proposed an approach that uses past defect data to improve reliability predictions. The study inferred the potential of requirement and design metrics to predict the number of faults at design stage of the development. The study had developed an early stage reliability predictor with the help of requirement and design level metrics. Numerical example was illustrated with both actual and simulated datasets. The analysis with example shows that the proposed approach works effectively in the early phases of software.

Cheung et. al (2008) developed a framework for reliability prediction of software components during the design phase. The study also emphasizes that product and process characteristics may be one of the mean to predict residual defects in the early stages. These characteristics are supposed to be embedded in software metrics. Therefore, software metrics may play a promising role for predicting the reliability of software while it is in its early phases of development. The authors highlighted the limitations of the study that the scalability of their reliability prediction technique at the system level remains a challenge and further investigation is needed.

Lahami et. al (2010) propose an incremental software development process that addresses reliability concerns, from early to late stages of development. Author has merge two dependability means: fault prevention and fault forecasting techniques in order to build reliable distributed software systems. Further the study has suggested that identification of suitable set of metrics should be considered a significant step in order to develop a defect prediction model with higher prediction accuracy. More recently, Hazra et. al (2013) presented formal methods for determining whether a set of components with given reliability certificates for specific functional properties are adequate to guarantee desired end-to-end properties with specified reliability requirements. Authors introduced a formal notion for the reliability gap in component based designs and demonstrate the proposed approach for analyzing this gap using a case study developed around an Elevator Control System.

Common problems with these existing approaches are: absence of applicability along with scalability, over-dependence on specific type of data and overlooking or neglecting quality of early stage documentation like, Software Requirements Specifications, which is the most critical document prepared as the requirements gathering, analysis and specifications have completed. Therefore, these approaches have become inappropriate or unsuited to provide trustworthy results.

Software reliability quantification has attracted immense interest from researchers as well as software practitioners since the early 1990's. Traditional methods for quantifying the software-reliability such as reliability growth models estimates

reliability on the basis of the defects observed during validation testing, where operational patterns represent how actually the product would be used (Lyu, 1996; Reibman and Veeraraghawan, 1991). However, quantifying software reliability in an early stage has been a tricky research topic that many researchers have attempted to resolve with limited success. Unfortunately, absence of failure data in early stages of software makes it challenging to measure the reliability. So there are just a few attempts, addressing the concept of early software reliability assessment or prediction (Jiang et al., 2007).

During the review of literature it is commonly observed that software metrics has been playing a prominent role in fault identification (Catal and Diri, 2009; Mizuno and Hata, 2009; Radjenovic et al., 2013). In a study (He et al., 2015) authors had focused on the selection of an appropriate metrics suite to develop a prediction model. The study also demonstrates that how this selection impact on the fault prediction accuracy. While, Maa et al., (2014), had proposed a reliability prediction model, that highlighted the potential of requirement and design metrics in defect prediction at the early stage of development. In another effort, the concept of bayesian networks was used by Yin et al, (2000) to publicize the relationship between software metrics and defect proneness.

In (Catal and Diri, 2009) authors supported the role of method-level metrics in predicting defects of application software. While the efforts had done by Radjenovic et al., (2013), drawn attention towards the potential of Chidamber and Kemerer's object-oriented metrics in predicting defects. The study also concluded that these

metrics are not only the most frequently used metrics but also used twice than other conventional metrics. Another work in the area of defect prediction (Pandey and Goyal, 2013) has considered the role of process maturity with software metrics, while developing a defect prediction model. The study had developed fuzzy profiles for different metrics followed by the fuzzy inference process, but the criteria behind these profiles were not justified properly.

In a study Georgieva, et al., (2011) have used the fuzzy logic approach for measuring software reliability, and concluded that participation of fuzzy logic has overcome the limitations of probability based reliability models. Another fuzzy based model proposed by Yadav et al., (2012) that predicts the residual faults during the testing stage. Another well known work done by Pandey and Goyal, (2010) where, a data mining technique (classification) was used with fuzzy logic to categorized software modules as fault prone or not.

While, the research (Aljahdali, 2011) had demonstrated, how fuzzy logic can solve the modeling issue of reliability? The study had developed a fuzzy based reliability growth model that estimate software defects at the testing stage during development. Khalsa, (2009) had also proposed an approach that make use of fuzzy sets to identify software modules with larger defect density in the design stage of SDLC. While Adaptive-Network based Fuzzy Inference System approach had been followed by Yaun and Zhang, (2011) to develop a reliability model based on Fuzzy-Neural hybrid network.

After briefly describing a variety of research works, some of the pertinent and resent studies have been described in the following paragraphs with proper detail followed by corresponding critical findings.

2.5.1 A Fuzzy Inference Model for Reliability Estimation of Component Based Software System

Jaiswal and Giri, (2015) developed a Reliability estimation model of component-based software system. Beside this author had also developed a model that computes reusability in terms of understandability, variability, portability, maintainability and flexibility.

Critical Findings:

- One important finding is the weight that the author had used for understandability, variability, portability, maintainability and flexibility to compute the reusability. All the five factors were multiplied by a fixed value (i.e. 0.2). This is not justified as each of these factors may have different magnitude of influence on reusability. Although, the study may use the multiple linear regression to get better values in this case.

- The study had not described the development as well as validation process properly. It is unclear how accurate the reliability prediction given by this approach would be.

- Author did not perform the correlation analysis among the initially identified factors, to discard those that provide redundant information (i.e. measures similar property). Applying Pearson's correlation test with suitable significance level can do this. For each couple of highly correlated factor, only one of them will be selected.

2.5.2 Reliability Estimation of Object-oriented Software: Design Phase Perspective

Kumar and Dhanda, (2015) proposed a Reliability estimation model of object-oriented software in design phase. The model computes reliability in terms of effectiveness and functionality. Prior to develop reliability model, study had developed separate models for effectiveness as well as functionality. All the three models have many serious technical issues that question on their validity as well as on the entire study itself.

Critical Findings:

- There are many factors that are more significantly impact on reliability than effectiveness and functionality. But overlooking them and considering effectiveness and functionality without any quantitative ground is not justified.

- The most critical point is that the equation of the developed reliability model (equation 4 in Kumar and Dhanda, 2015) shows that effectiveness and functionality negatively impacting the reliability value (i.e. both have negative coefficient), which is not true. Because each of these are positively correlated with the software reliability.

- Developing a model using just five records, questions on validity of the model.

- Similarly the significance (p value) of the 'encapsulation' in table 2 (i.e. 0.783), discourage its involvement in the 'effectiveness estimation model'.

2.5.3 Reliability Quantification of Object-Oriented Design: Complexity Perspective

Yadav and Khan, (2012a)**,** proposed and implemented a reliability quantification model for object-oriented design. Focus of the study was to compute complexity of object-oriented design, followed by reliability computation in terms of complexity. The study had used multiple linear regression to quantify complexity and reliability.

Critical Findings:

- The study had highlighted that the Inheritance impacted positively on complexity (means as inheritance increases the complexity of the OO design will also increases), but the proposed Complexity model (CEM) does not support this, as "IMc" has a negative coefficient that will impact on the OOD complexity inversely.

- The thesis had developed two multiple regression models (i) Complexity Estimation Model (CEM) and (ii) Reliability Estimation Model (REM).

- Similarly, the paper had also emphasizes that the Encapsulation is inversely proportional to the deign complexity (means as encapsulation increases the complexity of the OO design will go down), but the proposed Complexity model (CEM) does not support this, as "EMc" has a positive coefficient means the proposed model will increase the OOD complexity as the encapsulation increases.

- The study had not justified the goodness or statistical significance of neither of the model. It is not clear how efficiently these models are quantifying their respective dependent variables (Complexity and Reliability).

- In the Complexity Model the significance of individual independent variable was not shown, that is required to justify their participation as independent variables in the complexity model. (t Test should be used for this.)

2.5.4 Towards a Formal and Scalable Approach for Quantifying Software Reliability at Early Development Stages

Wende Kong, (2009) in his Ph.D**.,** proposed an approach to predict the reliability at the end of the requirements phase, on the basis of SRS document. Focus of the study was on the correctness and completeness of the SRS. The author had used the Cause-Effect Graph Analysis for predicting the reliability. The study mathematically formalized the cause effect graph, and applied it on SRS to identify its faults, subsequently fault tree was built through the identified SRS faults. In order to

analyze the fault tree Binary Decision Diagram (BDD) approach, along with an algorithm were used to quantifying the impact of the detected requirements faults on software reliability.

Critical Findings:

- The process of identifying SRS faults is totally manual, requires domain knowledge and understanding of the system under study along with inspector's creativity, experience and even intuition.

- Without prior and comprehensive knowledge of the system, the faults found through CEGA may not be correct and the final reliability estimation may not be very meaningful.

- Approach is very costly and time-consuming, specially, to construct an initial Cause Effect Graph (CEG) from a given informal specification.

- Not every aspect of a software system will be specifiable by a CEG, because a CEG can only capture functional requirements specified in the SRS. CEG analysis could not detect hidden requirements.

- Validation process was not up to the mark. It is unclear how accurate the reliability prediction given by this approach would be.

- Scalability is also one the issue, for large SRS it is very difficult to build and analyze the CEG.

2.5.5 Software Reliability Assessment Based on a Formal Requirements Specification

Hooshmand and Isazadeh, (2008) proposed an approach for early software reliability assessment based on software behavioral requirements. Viewcharts has been used to specify the behavior description of software systems. The author had also used the concept of Markov chain with viewchart, in order to determine the rate of system's transition among its different states. The study further predicated some states, for each of the system's view, those may cause system failures, and assess software reliability as the union of the probabilities of these failure states.

Critical Findings:

- Drawing viewchart from the system specification is a manual task and needs to be done by a person having proper awareness about the different dimensions of system's behavior.

- The study had not specified any rule or guidelines for drawing the viewchart specification from the corresponding system behavior.

- To calculate the rate of system state transition, a prototype of the system needs to be build based on its viewchart specifications. Besides that the prototype will be executed with some input from the corresponding operational profile. This makes the approach quite complicated and expert specific, especially at the requirement stage.

- The approach also has the scalability issue, for systems of significant size developing the viewchart specification would be a challenging job.

- As the reliability assessment is totally based on the union of the probabilities of failure states, therefore for each of the view identifying and introducing the probable events those may cause a system failure, needs the comprehensive knowledge about the different behaviors of the system.

2.6 SUMMARY OF REVIEW FINDINGS

After reviewing a variety of studies on Reliability quantification, it is the time to sum up the findings, and suggest the way to reach to a feasible solution. Following is the summary of critical observations noticed during the review:

- No consensus or standard steps/procedure among researchers for predicting software reliability.

- Most of the studies that incorporated Multiple Linear Regression had not bothered about multicollinearity and autocorrelation at all.

- Some of the studies have been suffering from severe technical shortcomings that compel to deduce that those researchers had not put their sincere effort.

- Dataset used for empirical analysis were inappropriate in size and also lacks quality data.

- Some researchers performed quantification quite well, but did not provide suggestive measure and guidelines to be followed for controlling the unreliability.

- 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.

2.7 CONCLUSION

To provide a foundation the chapter starts with the brief taxonomy of software reliability measurement models, followed by emphasizing the importance of early reliability prediction. After presenting the comprehensive state of the art, summary of review finding concludes the chapter. Before ending this chapter, it is needed to suggest a solution that will overcome the problems highlighted in the chapter. Therefore, in the next chapters the researcher is going to present a roadmap in the form of a prescriptive framework followed by its implementation and validations.