CHAPTER 2

LITERATURE SURVEY
LITERATURE SURVEY

2.1 SOFTWARE QUALITY AND RELIABILITY

Building high-quality and high-reliability software is difficult. The need to develop high-quality complex software continues to drive research in a number of areas in software engineering. Software engineering has been the fastest developing technology during the last century. However even now “there is no complete, scientific measure to assess software quality and reliability”. Software reliability assessment received much impetus from this requirement and many models are developed to assess software [136]. With the rapid development of computer hardware, the bottleneck of system quality lies in its software. Therefore, people are looking forward to reaching such a situation where software works with high reliability, maintainability and performance, reduced turnaround time, and low development and maintenance cost [139].

One approach to solve the above problem effectively is to predict the quality of software in the early stages of the software development process [114]. For an established software company, there are often data collected from past projects or releases. They are useful for early prediction of software quality. Early prediction, i.e., predicting the quality software modules prior to software testing and operations, benefits software development teams from many aspects [131]. On the one hand, it will be possible to estimate the impact the errors will exert on the final software product, and such timely quality estimation can be used to direct cost-effective quality improvement efforts to the high-risk modules, so that in advance, the quality of the software can be controlled. On the other hand, it is helpful to find the ‘best’ software design out of numerous options,
namely to find the design that will have the highest software quality among a large number of feasible software designs. Above all, it is obvious that this approach of predicting the quality of software in the early stages of the development process helps in ensuring the final quality of software product, shortening the development cycle and reducing the cost of development and maintenance. Developers and customers can come up with realistic overall requirements for the target system early, avoiding possible expensive rework in later stages of the software lifecycle [142].

There are many papers advocating statistical models and metrics which purport to answer the quality question. Defects, like quality, can be defined in many different ways but are more commonly defined as deviations from specifications or expectations which might lead to failures in operation [50]. Generally, efforts have tended to concentrate on the following three problem perspectives:

- predicting the number of defects in the system
- estimating the reliability of the system
- understanding the impact of design and testing processes on defect counts and failure densities

A wide range of prediction models have been proposed. Complexity and size metrics have been used in an attempt to predict the number of defects a system will reveal in operation or testing. The historical measurement data is used to build quality estimation models. The quality of these models in turn depends heavily on the quality of the sample used.
2.2 SOFTWARE RELIABILITY MODELLING

The main reason for a worldwide growing interest in software reliability and thus software quality are basically due to the following reasons:-

- Software is becoming central to many life safety critical systems
- Software is created by error prone human beings
- Software is executed by error intolerant machines
- Very often software development is dictated by budget and schedule rather than by a concern for quality and reliability [113][136].

Software reliability has been an active research area since early seventies and many software reliability models have been introduced. Software reliability has been defined as the probability that a computer program will do its intended operation for a specified time in a specified environment. When a software fails to do its intended function, it is called a failure. Thus failure means the program in its functioning has not met user requirements in some way.

Applying the available software reliability models is commonly adopted to find the reliability of the software. Most reliability models attempt to estimate the reliability of the software based on its error history either during its debugging phase or during its validation phase. The assessment of software reliability is important right from the development phase of software life cycle. A number of models have been proposed in the literature for characterizing (measuring, estimating and predicting) software reliability [99][100].
A proliferation of software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. None of the models can capture a satisfying amount of the complexity of software; constraints and assumptions have to be made for the quantifying process. Therefore, there is no single model that can be used in all situations. No model is complete or even representative. Most software models contain the following parts: assumptions, factors, and a mathematical function that relates the reliability with the factors. The mathematical function is usually higher order exponential or logarithmic [36].

2.2.1 Software Reliability Prediction Models

Software modeling techniques can be divided into two subcategories: prediction modeling and estimation modeling. The prediction of software reliability can determine the current reliability of a product, using statistical techniques based on the failures data obtained during testing or system usability. When prediction model uses historical data, the model is made prior to development or test phases and can be used as early as concept phase and it predicts reliability at some future time. Representative prediction models include Musa's Execution Time Model, Putnam's Model. Using prediction models, software reliability can be predicted early in the development phase and enhancements can be initiated to improve the reliability [99]. The premise of most prediction models is that the failure rate is a direct function of the number of faults in the program and that the failure rate will be reduced (reliability will be increased) as faults are detected and eliminated during test or operations. This premise is reasonable for the typical test environment and it has been shown to give credible results when correctly
applied. However, the results of prediction models will be adversely affected by the changes in failure criteria, changes in code under test and changes in computing environment.

2.2.2 Software Reliability Estimation Models

Estimation model uses current data, usually after some testing has been done and data is collected and estimates the reliability at present or future. Representative estimation models include exponential distribution models, Weibull distribution model, Thompson and Chelson's model, etc. Exponential models and Weibull distribution models are usually named as classical fault count/fault rate estimation models, while Thompson and Chelson's model belong to Bayesian fault rate estimation models. Some of the ways to develop a estimation model is by describing it as a stochastic process or defining the probability density or distribution function by specifying the hazard function [36]. There are the following three general classes of software reliability estimation models: exponential non-homogeneous Poisson process (NHPP) models, non-exponential NHPP models, and Bayesian models. The exponential NHPP models use the stochastic process and the hazard function approach. Representative models in this class include Schneidewind’s model, Shooman’s model, Musa’s basic model, Jelinski and Moranda’s model. Non-exponential NHPP models also use the stochastic process and the hazard function approach. They are applicable after completion of testing. Early fault corrections have a larger effect on the failure intensity function than later ones. Representative models in this class include binomial and Poisson models, Yamada’s s-shaped model.
and Musa and Okumoto’s logarithmic Poisson model [98]. Bayesian models assume that the hazard function is directly proportional to the number of faults in the program, and hence the reliability is a function of this fault count. In the Bayesian approach [33] a program can have many faults in unused sections of the code and exhibit a higher reliability than software with only one fault in a frequently exercised section of code. Representative models of this class are those developed by Littlewood [113].

Software reliability practitioners have applied reliability prediction models during several stages of a software development project-design and coding, integration and testing, and after release. Applying reliability growth models during design and coding will not give acceptable results because the software is often unstable and does not contain all the required functionality. Making reliability predictions after release in the field is possible, but high quality data is difficult to collect, and it becomes too late for effective project management.

2.3 **SOFTWARE DEFECTS**

As stated earlier software defects can be defined as imperfections in the development processes making the software fail to meet the specified requirements. Defects get injected at all phases of development, but the effect of defects in requirements is more severe than a coding defect. In a dynamic (operational) environment, some problems may be caused by *failures*. In a static (non-operational) environment, such as a code inspection, some problems may be caused by defects. In both dynamic and static environments, problems also may be caused by misunderstanding, misuse, or a number of
other factors that are not related to the software product being used or examined. Project procedures like configuration management, incident logging, documentation, and standards should help to reduce the likelihood of defects.

2.4 SOFTWARE DEFECT PREDICTION MODELS

A wide range of defect prediction models are proposed [10]. To establish and maintain control over the development and maintenance of a software product, it is important that the software developer and maintainer measure software problems and software defects found in the software product to determine the status of corrective action, to measure and improve the software development process, and to the extent possible, predict remaining defects or failure rates [12]. By measuring problems and defects, we obtain data that may be used to control software products.

It is equally clear that the number and frequency of problems and defects associated with a software product are inversely proportional to the quality of the software. Software problems and defects are among the few direct measurements of software processes and products. Such measurements allow us to quantitatively describe trends in defect or problem discovery, repairs, process and product imperfections, and responsiveness to customers. Problem and defect measurements also are the basis for quantifying several significant software quality attributes, factors, and criteria for reliability, correctness, completeness, efficiency, and usability among others. Many organizations want to predict the number of defects (faults) in software systems before they are deployed, to gauge the likely delivered quality and maintenance effort [45]. To help in this, numerous software
metrics and statistical models have been developed. The models are weak because of their inability to cope with the as yet unknown relationship between defects and failures. There are fundamental statistical and data quality problems that undermine model validity. More significantly, many prediction models tend to model only part of the underlying problem and do not specify it correctly.

Validating these models is also difficult. Theoretical validation and empirical validation are conducted to validate the models. Theoretical validation ensures that a product metric is a proper numerical characterization of the property it claims to measure. Empirical validation demonstrates that a product metric is associated with some important quality attributes. Most of the predictive models are validated using different sample data and various dependent and independent variables. Predictive modeling techniques are only as good as the data they are based on. The relationship between software product measures and the presence of faults cannot be considered an assumption that holds good for any data set and project. This makes it difficult for practitioners to decide which model is trustworthy [114].

2.4.1 Prediction using size and complexity metrics

Early defect prediction studies are based on size and complexity metrics. The studies were about prediction of number of defects and the relationship of software metrics with errors. Size metrics were used as both a control metric predictor and quality predictor. Large components are expected to have a larger number of residual errors, and are more difficult to understand than smaller components and thus size has an impact on reliability.
Another early study argued that the expected number of defects increases with the number \( n \) of code segments; a code segment is a sequence of executable statements which, once entered, must all be executed. Specifically the theory asserts that for smaller number of segments, the number of defects is proportional to a power of \( n \); for larger number of segments, the number of defects increases as a constant to the power \( n \). [50].

Halstead proposed a number of size metrics, which have been interpreted as ‘complexity’ metrics, and used these as predictors of program defects. McCabe’s complexity metric was also used to predict the probability of introducing errors. McCabe’s cyclomatic complexity, has been used in many studies, but it too is essentially a size measure (being equal to the number of decisions plus one in most programs). It was observed that there is an optimum size for the code. It was dubbed as the ‘Goldilocks Principle’ with the idea that there is an optimum module size that is “not too big or too small”. The realization that size-based metrics alone are poor general predictors of defect density spurred on much research into more discriminating complexity metrics.

### 2.4.2 Prediction using testing metrics

Some of the most promising local models for predicting residual defects involve very careful collection of data about defects discovered during early inspection and testing phases. The idea is very simple: you have \( n \) pre-defined phases at which you collect data \( d_n \), the defect rate. Suppose phase \( n \) represents the period of the first 6 months of the product in the field, so that \( d_n \) is the rate of defects found within that period. To predict \( d_n \) at phase \( n-1 \) (which might be integration testing) you look at the actual sequence \( d_1, \ldots, d_{n-1} \).
and compare this with profiles of similar, previous products, and use statistical extrapolation techniques [118]. With enough data it is possible to get accurate predictions of $d_n$ based on observed $d_1...d_m$ where $m$ is less than $n-1$. The IBM NASA Space shuttle team is achieving similarly accurate predictions based on the same kind of approach.

One class of testing metrics that appear to be quite promising for predicting defects are the so called test coverage measures. A structural testing strategy specifies that we have to select enough test cases so that each of a set of "objects" in a program lie on some path (i.e. are ‘covered’) in at least on some test case. For example, statement coverage is a structural testing strategy in which the "objects" are the statements. For a given strategy and a given set of test cases we can ask what proportion of coverage has been achieved. The resulting metric is defined as the Test Effectiveness Ratio (TER) with respect to that strategy. Clearly we might expect the number of discovered defects to approach the number of defects actually in the program as the values of these TER metrics increases. Veevers & Marshall report on some defect and reliability prediction models using these metrics which give quite promising results.

2.4.3 Prediction using process quality data

There are many experts who argue that the ‘quality’ of the development process is the best predictor of product quality (and hence, by default, of residual defect density) [50]. There is a dearth of empirical evidence linking process quality to product quality. The simplest metric of process quality is the 5-level ordinal scale SEI Capability Maturity Model (CMM) ranking.
The CMM (Capability Maturity Model) differentiates among five levels of software-process maturity: initial, repeatable, defined, managed, and optimizing [58]. For each level, it defines key process areas, with each area containing key practices that must be performed to rate that particular maturity level. A key practice specifies key indicators, which directly relate to at least one question. Thus, while the CMM provides useful guidelines for any organization wanting to find out what has to be done to reach the next maturity level, its use of a single maturity measure for an entire process does not sufficiently support a quantitative analysis of a process's strengths and weaknesses.

There are a number of interesting observations about the way process complexity metrics are used to predict defect counts:

• The models ignore the causal effects of programmers and designers.

• Overly complex programs are themselves a consequence of poor design ability or problem difficulty. Difficult problems might demand complex solutions and novice programmers might not produce good code.

• Defects may be introduced at the design stage because of the over-complexity of the designs already produced. Clerical errors and mistakes will be committed because the existing design is difficult to comprehend.
2.4.4 Prediction using Bayesian Belief Networks

Bayesian Belief Networks allow us to express complex inter-relations within the model at a level of uncertainty, which commensurate with the problem [50]. Clearly the ability to use Bayesian Belief Networks to predict defects will depend largely on the stability and maturity of the development processes. Organizations that do not collect metrics data, or follow defined life-cycles or perform any forms of systematic testing will never be able to build or apply such models. Similarly, replication of experimental results can only be predicated on software processes that are defined and repeatable.

2.4.5 Earlier Models and their shortcomings

The software lifecycle consists mainly of the following phases: requirement and specification, design, coding, testing and operation/maintenance. During the testing phase, the software programs are tested and detected faults are corrected. This is the most important phase of software development because of the high cost associated with it. Conventional software reliability models are very useful in analyzing collected failure data and assessing the reliability level at that point of time. Mostly they come into picture after the testing is completed. Predicting the total number of defects before testing begins, has an impact on the quality of the product being delivered. Also ideally, any model using defects found in one phase of verification to predict defects found in subsequent phases should incorporate causal factors such as verification/testing effort and process quality. Defects are caused by many factors- in reality the propensity to introduce defects will be influenced by many factors unrelated to code or design complexity. There are a number of causal factors at play when we want to explain the presence of defects in a
Eliciting requirements is a difficult process and is widely recognized as being error prone. Defects introduced at the requirements stage are claimed to be the most expensive to remedy if they are not discovered early enough. When assessing a defect, it is useful to determine when it was introduced. Broadly speaking there are two types of defect; those that are introduced in the requirements and those introduced during design (including coding/implementation which can be treated as design). A defect model is useful if it can explain why a module has a high or low defect count and if we are able to learn from its use, which are the error prone areas, otherwise we could never intervene and improve matters.

But for mission critical software like flight software making a reliability assessment based on failure data collected during testing phase alone is very late. If a defect detected during testing is traced to a requirement error, all the steps of requirement analysis, design, coding, testing have to be repeated. This will have a serious impact on schedule and cost. Hence in flight software, detailed verification is done in every phase to detect the defects and correct them before proceeding to the subsequent phase. This way most of the defects are eliminated before the testing phase. Very few errors are detected during testing and they are mainly system dependent or performance dependent errors.
Hence a quality and reliability model which can predict the quality utilizing the defects detected at different phases is required. None of the models currently available do that. In fact none of the available quality models nor the statistical reliability prediction models are able to predict the number of defects that are likely to occur based on the conditions in which the software is developed. The metrics based models are incapable of predicting defects accurately using size and complexity metrics alone. Furthermore, these models offer no coherent explanation of how defect introduction and detection variables affect defect counts. The quality of the software depends to a great extent on the environment in which it is developed and tested. None of the existing models, be it statistical model or metrics based model have features to feed the parameters that play a major role in the quality of the flight software like quality of documentation, domain expertise of programmer, requirements traceability etc into the model. Hence alternate method of predicting errors and hence assessing the quality of flight software was explored.