CHAPTER 1

INTRODUCTION

Software development is a process of finding and developing a product for the particular purpose in different level. It is required to take a more concern on the software development process to implement the better software that can provide the essential services to the people. Software engineering is an process of developing and providing the product that can satisfy the user requirements in various levels.

1.1 CISQ'S QUALITY MODEL

Even though “quality is a perceptual, conditional and somewhat subjective attribute and may be understood differently by different people” (as noted in the article on quality in business), software structural quality characteristics have been clearly defined by the Consortium for IT Software Quality (CISQ). Under the guidance of Bill Curtis, co-author of the Capability Maturity Model framework and CISQ's first Director; and Capers Jones, CISQ's Distinguished Advisor, CISQ has defined five major desirable characteristics of a piece of software needed to provide business value (CISQ software quality, 2009). In the House of Quality model, these are “Whats” that need to be achieved.
1.2 SOFTWARE QUALITY

In the context of software engineering, software quality refers to two related but distinct notions that exist wherever quality is defined in a business context:

Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product (Pressman Roger S, 2005).

Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability, the degree to which the software was produced correctly.

Structural quality is evaluated through the analysis of the software’s inner structure, its source code, at the unit level, the technology level and the system level, which is in effect how its architecture adheres to sound principles of software architecture outlined in a paper on the topic by OMG (2013). In contrast, functional quality is typically enforced and measured through software testing.

In computer technology, a bug is a coding error in a computer program. (Here we consider a program to also include the microcode that is manufactured into a microprocessor.) The process of finding bugs before program users do is called debugging. Debugging starts after the code is first written and continues in successive stages as code is combined with other units of programming to form a software product, such as an operating system.
or an application. After a product is released or during public beta testing, bugs are still adopt to be discovered. When this occurs, users have to either find a way to avoid using the “buggy” code or get a patch from the originators of the code. Bugs are the misery for any programmer, hence debugging them is part of the program development process. The effective debugging of Java requires some experience.

A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most bugs arise from mistakes and errors made in either a program's source code or its design, or in its components and operating systems used by such programs. A few are caused by compilers producing incorrect code. A program that contains a large number of bugs, and/or bugs that seriously interfere with its functionality, is said to be buggy (defective). The software industry has put much effort into reducing bug counts.

Several schemes assist managing programmer activity so that fewer bugs are produced. Software engineering (which addresses software design issues as well) applies many techniques to prevent defects. For example, formal program specifications state the exact behavior of programs so that design bugs may be eliminated. Unfortunately, formal specifications are impractical for anything but the shortest programs, because of problems of combinatorial explosion and indeterminacy.

Unit testing involves writing a test for every function (unit) that a program is to perform.

In test-driven development unit tests are written before the code and the code is not considered complete until all tests complete successfully.
Agile software development involves frequent software releases with relatively small changes. Defects are revealed by user feedback.

Open source development allows anyone to examine source code. A school of thought popularized by Eric S. Raymond as Linus's Law says that popular open source software has more chance of having few or no bugs than other software, because “given enough eyeballs, all bugs are shallow”. This assertion has been disputed, however computer security specialist Elias Levy wrote that “it is easy to hide vulnerabilities in complex, little understood and undocumented source code,” because, “even if people are reviewing the code, that doesn't mean they're qualified to do so.”

Tools for code analysis help developers by inspecting the program text beyond the compiler's capabilities to spot potential problems. Although in general the problem of finding all programming errors given a specification is not solvable, these tools exploit the fact that human programmers tend to make the same kinds of mistakes when writing software.

Tools to monitor the performance of the software as it is running, either specifically to find problems such as bottlenecks or to give assurance as to correct working, may be embedded in the code explicitly (perhaps as simple as a statement saying print“I AM HERE”), or provided as tools. It is often a surprise to find where most of the time is taken by a piece of code, and this removal of assumptions might cause the code to be rewritten.

Software testers are professionals whose primary task is to find bugs, or write code to support testing. On some projects, more resources may be spent on testing than in developing the program. Measurements during testing can provide an estimate of the number of likely bugs remaining, this becomes more reliable the longer a product is tested and developed.
1.3 DETERMINING THE TYPE OF ERROR

There are three general types of errors that occur while playing a JavaScript program. These errors are:

- Load-time errors
- Runtime errors
- Logic errors

1.3.1 Load-time errors

Load time errors are caught by JavaScript as Navigator loads the script. This error stops the script from functioning before it has a chance to start. During the loading process, JavaScript spots serious errors that can cause the script to fail. The page has been successfully loaded for the script. Load-time errors are commonly caused by problems in syntax. The error may be located at a different section of the line or even on another line.

1.3.2 Runtime errors

These are errors that occur when the script is actually playing. The nature of the error will be specified, along with a line number that can always be inaccurate. The load-time errors are generally caused by mistakes in syntax, runtime errors are most often due to improper use of commands. Another type of runtime error occurs when JavaScript's objects are misapplied.
1.3.3 Logic error

Logic error occurs when the script does something different than that suggested by logic. Misplaced parenthesis or misapplied statement cannot cause errors, but rather to a mistake in the way the script is constructed. Script fails when the data is not in the proper format.

Different types of tools are employed to detect the bugs. The bug detection tools are functioned to detect the known types of bugs and hence different tools detect different types of bugs and also exempt some types of bugs. Thus makes this research work necessary to analyze the different types of bug detection tools.

Predicting and fixing the fault in the software is difficult, and it needs some significant effort. The software fault can cause series problems, which arises due to the lack of proper planning. They are liable to fault and the time taken to debug and fix them is more than the actual development time. Zero defect products are a dream of a software engineer. For producing good quality software the companies are spent 60% of their production cost. They were concentrating on testing the product, yet many faults still exist. This leads to unpredictable behavior of the software and sometimes becoming terrible or unusable.

Typically a software is a product that contains known and unknown defects. This leads to a continuous process of developing software by removing known faults and uncovering new defects over time. As new faults are uncovered, highest priority is given to the fixing of bugs which is a major task for the software development team.
Many programs suffer from performance bugs, source code locations in which a simple change can significantly increase the speed of the program. Addressing such problems requires to identify bottlenecks of the performance, careful study of the potential root causes, and to experiment with code transformations until one is found that leads to a significant performance improvement. Because this process is time-consuming and cumbersome, developers often rely on compiler optimizations that however, are limited to a predefined set of transformations that can certainly be applied for preserving the semantics. This situation suggests that automatic detection technique and fixing bugs will be of considerable benefit.

Several works has been proposed to analyze the bugs in the system. Work has been done in a way to prioritize the bugs with respect to the crash report which has occurred in different scenarios. Prioritizations of bugs are identified with the crashes that may happen due to the corresponding bug. Analysis was made considering the parameters that can lead software to failure due to large amount of crashes in the system. Analysis of the concurrent bugs is done with the help of finding the similarity among the various types of bugs in the network. Because of different formats of bugs, it will be more difficult to identify the concurrent bugs. This is done by finding the synchronization among the different bugs.

There are multiple ways to construct the software without errors and to construct the good base software was discussed. Various studies were carried in terms of software availability and the software requirements that are elicited. Efficient software can be developed with the help of uniqueness of software characteristics. Here security level is defined in terms of the software failure which will occur at the compilation time of software.
Ways for building secure and reliable software with the consideration of the software development is also discussed. Reliable software development can be obtained with the consideration of the requirements which are gathered from the different phases in terms of previous history of similar type of software. This method results in an effective way for constructing the software with more reliability without software failure. Also symbolic execution parameters were developed for the JavaScript in web pages Saxena (2010).

Approaches were also introduced to gather software requirements from various phases that intend to produce good quality software. However it is difficult to get the software requirements that are more relevant to the software which is yet to be developed. Another technique was introduced to produce more secured software which can lead to a successful completion of software development. The agile methodology leads to a highly effective implementation of the software development based on the iterative and incremental software development.

Code review, unit testing, system testing and integration testing are the traditional process to identifying faults. It also includes a software fault detection methods that accompanies software evaluation and software design processes. The software tester records the identified faults during evaluation and testing process. The well organized software development also analyzes faults using statistical processes, systems like Six Sigma are used effectively to reduce defects.

Reliability assessment on updating the existing systems is a major issue in software industry. Existing systems need to be constantly maintained, inorder to add new components and to recover from fault, hence there is saddle in updating. Reliability assurance after an upgrade is assessed by
observing failure behavior of the upgraded system in operation with the appropriate knowledge of the existing systems.

Traditionally software metrics are used to define the software complexity, to estimate programming time. Extensive research has also been carried out to calculate the number of defects in a module using software metrics. Software metrics are used for obtaining quantitative measurements of the software or its specifications. Mostly value of the metrics is represented as an interval, ordinal or nominal scale. Metric values in ratio scale are preferred for the mathematical equations used in software process as it allows for easy and meaningful computation.

The software metrics is a widely researched area. The first and most extensive report of software metrics program containing key guidelines forming a base for subsequent metrics program was done by Grady and Caswell. Goal Question Metric (GQM) was proposed, by Basili and Rombach, using ideas from the total quality management to ensure that the metrics activities are always goal driven. The GQM ensured that the metrics program should collect only those metrics which are relevant to a particular goal. The works of Briand et al., 1996 highlights the need of scale types to define measurement for software metrics activities.

An emerging approach for fault prediction is the use of various techniques to predict the problematic areas in the software. These techniques extensively reliability on software metrics associated with the system. Bayesian inference under simplifying assumptions formed the base of the reliability assessment studies. The accuracy of predictions achieved due to reliability is discussed.
Fault prediction models are used to find the reliability of the system, estimate the efficiency of the design and testing process over a number of defects. The byproduct of the above process is the reduction of staffing required and the maintenance cost. Fault prediction models use machine learning methods to learn and predicting potentially defected modules within the software. Cyclomatic complexity and size of the software are the most commonly used metrics for faulty models. The testing metrics that are created in the test phase prove to be useful to estimate the sequence of faults. Metrics from all the steps of the software development life cycle such as design, implementation and testing should be utilized and connected to specific dependencies. For designing an effective prediction model, metrics from a single process will not be sufficient.

1.4 SOFTWARE QUALITY ASSURANCE

Schumeyer and McManus have defined software quality as “the fitness for use of the total software product”. Good quality software does exactly what it is supposed to do and is interpreted in terms of satisfaction of the requirement specification laid down by the user.

1.4.1 Quality Assurance

Software quality assurance is a methodology that determines the extent to which a software product is fit for use. The activities that are included for determining software quality are:

- Auditing
- Development of standards and guidelines
- Production of reports
- Review of quality system
Quality Factors

Correctness: Correctness determines whether the software requirements are appropriately met.

Usability: Usability determines whether the software can be used by different categories of users (beginners, non-technical, and experts).

Portability: Portability determines whether the software can operate in different platforms with different hardware devices.

Maintainability: Maintainability determines the ease at which errors can be corrected and modules can be updated.

Reusability: Reusability determines whether the modules and classes can be reused for developing other software products.

1.5 OBJECT-ORIENTED METRICS

Metrics can be broadly classified into three categories: project metrics, product metrics, and process metrics.

1.5.1 Project Metrics

Project Metrics enable a software project manager to assess the status and performance of an ongoing project. The following metrics are appropriate for object-oriented software projects:

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Number of subsystems
1.5.2 Product Metrics

Product metrics measure the characteristics of the software product that has been developed. The product metrics suitable for object-oriented systems are:

**Methods per Class:** It determines the complexity of a class. If all the methods of a class are assumed to be equally complex, then a class with more methods is more complex and thus more susceptible to errors.

**Inheritance Structure:** Systems with several small inheritance lattices are more well-structured than systems with a single large inheritance lattice. As a thumb rule, an inheritance tree should not have more than 7 (± 2) number of levels and the tree should be balanced.

**Coupling and Cohesion:** Modules having low coupling and high cohesion are considered to be better designed, as they permit greater reusability and maintainability.

Response for a Class: It measures the efficiency of the methods that are called by the instances of the class.

1.5.3 Process Metrics

Process metrics help in measuring how a process is performing. They are collected over all projects over long periods of time. They are used as indicators for long-term software process improvements. Some process metrics are:

- Number of KLOC (Kilo Lines of Code)
- Defect removal efficiency
• Average number of failures detected during testing
• Number of latent defects per KLOC

1.6 PROBLEM SPECIFICATION

The problem that are found at the time of software development process are

➢ The software might lead to wrong output due to erroneous occurred in some modules at the time of implementation
➢ The software can be developed wrongly because of fault occurred at the design phase
➢ Bug present in the particular module would affect entire software due to its coupling and coherence properties
➢ It would be very difficult to predict and find the fault after development of software due to presence multiple modules and inter linkage functions

1.7 OBJECTIVE

The research objective is to present a novel software framework for the reliable and efficient prediction of faults that are present in the OOP programming language. In this research work, the overall impact of the dependent and interdependent modules present in the software that shares their field and information along with each other for completing the task is analyzed and based on which faults are predicted without affecting any dependent modules. The fault prediction is done with the consideration of OOP metrics and the various software dependency modules.
Designing developing and implementing a good quality product requires efficient measures to precisely monitor the internal quality attributes, like coupling, cohesion, size and complexity, throughout the software development life cycle. Software metrics have been extensively and successfully used to measure the internal quality attributes for object oriented software systems. Such internal quality attributes in turn give a measure of the external quality attributes like maintainability, modifiability, understandability, reusability, and testability etc Chidamber et al (1994), Briand et al (1996), Briand et.al.(2000).

UML model provides adequate information at the analysis phase for evolution of software metrics. Existing works provide the estimation based on the UML diagram and do not make use of the information available through the UML documentation. In addition to this, they do not adhere to any standard. Hence, the technology independent of Function Point Analysis and dependent UML Class Dependency Diagrams are combined to provide another approach for finding the relationship between the classes.

The work presented in this Thesis also investigates the existing relationship between software metrics and development effort. This approach uses fault detection in the Classes using UML diagrams, fault prediction schemes at code level and fault classifications using ANFIS model. The main conclusion of this research to provides the software quality and the ways that can be adapted for the assuring the software quality.
1.8 RESEARCH CONTRIBUTION

The major contribution of the present research works are

- Predicting fault using Java reflection which measures the coupling and cohesion metrics of OOP languages
- Predicting fault prone classes at the design level using UML class diagrams
- Predicting and classifying faults using the hybrid ANFIS prediction model

![Proposed framework](image)

Figure 1.1 Proposed framework
1.9 ORGANIZATION OF THESIS

Chapter 1 provides the overviews of the software quality, object oriented programming language and the role OOP in java language. This chapter mainly concentrates to analyze the bugs that are present in the object oriented system and their detection process.

Chapter 2 provides the brief discussion of related work on various bug detection schemes and the schemes that assures the software quality in terms of prediction of software bugs.

Chapter 3 provides the detailed description of fault prediction scheme on the java language with the concern of coupling and cohesion metrics.

Chapter 4 discusses the fault detection in the Classes at the UML design level.

Chapter 5 provides the detailed view of the classification process that is performed on the object oriented system faults.

Chapter 6 discusses about the result and discussion of the overall research work and analyzes the performance of the proposed approaches.

Chapter 7 concludes of the various fault prediction scheme performance at the design level and the coding level with future direction.
Chapter 1 provides the summary about the detailed overview of the software development and the causes of bugs in the software development process. It provides the short summary about the software quality and the ways that can be adapted for the assuring the software. And also this chapter provides the summary of the different types of possible bugs which might be present in the object oriented system.