CHAPTER 1

INTRODUCTION

Among various popular research areas, Object oriented measurement is also one, which is confirmed by the suggested works in the following literatures, they are: i) Lot of frameworks for coupling and cohesion (ii) A huge number of various measures for object oriented attributes like coupling, cohesion, and inheritance. But it has lot of negative views, where the object oriented measures were established. As neither a standard terminology nor formalism prevails, various measures were indicated in uncertain way which restricts their usage. These features, makes it complex to get the knowledge about how the different measures communicate each other. For instance, it have various decisions, that define a coupling measure and these methods should be created according to the target of the measure and by determining an empirical model which works on hypotheses. Sadly, the measures in the work doesn’t have encouragement for these decisions and also this makes it a complex task to understand the fundamental consideration of the measure, For instance it is uncertain about establishing a coupling connection among two classes or what is the strength of the coupling connection measured, when it is distinguished with the varieties of connections and which utilizes the other coupling measures. So these results in an uncertain condition of what the potential uses of the current measures are and how its different coupling measures can be utilized in a complementary
manner. This acquire more detailed picture of the coupling in an object-oriented systems.

1.1 OVERVIEW

In software development life cycle, Fault prediction is an important step to minimize the failures and is proceed at the time of the initial planning to recognize the fault-prone modules. Fault prediction not only provides an quality of monitoring at the time of developing the software, but also it gives an important information to proceed an appropriate verification and validation approaches, which in turn it leads to the enhancement of the efficiency and effectiveness of fault prediction. The effectiveness can be acquired by enforcing the earlier data based on the faults and estimates its performance against other part of the fault data. Various researchers focus on constructing the prediction models for software fault prediction but it gives less importance for the efficiency of the fault prediction.

Recently, software development preferred to work on the Object-Oriented (OO) paradigm. With the help of the software metrics, the quality of software can be improved. Several metrics were given by the researchers and practitioners to compute the quality of software. Abreu MOOD metric suite (Abreu and Carapuca 1994) is the one of best example for software quality metrics.

The quality attributes of software such as effort and fault pronenessness can be checked by these metrics. These metrics help to estimate the developed software’s quality. In general, the FURPS is an acronym representing a model for classifying software quality attributes, it have Functionality, Usability, Reliability, Portability, and Supportability. This model mainly concentrates different perspective to enhancing the reliability as
well as minimizing the faults in the software. Different conventional methods are available in literature to predict the reliability of a class. But, the machine learning techniques doesn’t have much concentration about the fault prediction. A subset of machine learning methods is the artificial intelligence techniques, which has the capability of computer, software and firmware to compute the properties of a class that human beings can identify as intelligent behaviour. So, these methods have the ability to approximate the non-linear function with more precision. So, these can be enforced for quality estimation to accomplish good accuracy.

1.2 OBJECT ORIENTED SYSTEM

The basic OOP principles are well-known by the ASP.NET developers, as this knowledge is important while developing the fault prediction software. For development of client-side code which uses the Microsoft AJAX Library and JavaScript's OOP features. Understanding these features is a great dispute, as JavaScript's OOP model varies from that of languages like #, VB.NET, C++, and Java.

The World Wide Web content production and the web page development, OOPS (Yanget al. 1999) concept is utilized. OOPS provides constructors and prototyping for sharing the functionality of codes in the JavaScript. At runtime, the objects can be changed by utilizing the adding properties or updating the methods in OOPS and its user friendliness makes it used broadly. In OOPS concept the web elements were assumed as objects, which varies from java programming language where Java is real time programming language but JavaScript cannot be written as separate coding because it gives the route for logical communication to the web pages. HTML language gives the user friendly flexible environment to the users, which has JavaScript where the web page information can be utilized. Java script enables
quick response to the users of offline stand-alone applications like site specific browsers and desktop widgets. It is generally utilized as addition of client side behaviour to the HTML web pages dynamically.

OOPS is an object-based language. As similar to, C# creates an object, call their methods, and pass them as parameters, and so on. Nevertheless, JavaScript is not considered as a fully object-oriented language, since, it has shortage in support for few features which can be identified in real OOP languages or it simply executes them in a different manner. Here, it has a list of concepts, which is used more frequently.

- Object
- Class
- Encapsulation
- Aggregation
- Reusability/ Inheritance
- Polymorphism

1.2.1 Objects

In Object Oriented Programming (OOP) objects were essential. An object is a real world entity which is the support for programming language. It can be a real-life object or complicated ideas. The characteristics of the object were termed as properties in OOPS and the actions were termed as methods.
1.2.2 Classes

In real life, similar objects can be gathered according to their criteria. In OOPS, a class is a blueprint or recipe for an object. An object is also termed as instance. Developing various objects with the help of similar classes can be performed since a class is considered as a template, while the objects are concrete instances according to the template.

The JavaScript and the "classic" OO languages have common difference. In JavaScript, it has no classes and everything works on objects. JavaScript has the prototype’s view, which is also considered as an object.

1.2.3 Encapsulation

Encapsulation, explains that an object has both data and methods.

The encapsulation is known as information hiding. Consider an object where the implementation of the interface is hidden from users. Similar things happen in OOP, when the code utilizes an object by calling its methods. The code doesn't require any knowledge about how the methods work internally. Whereas, JavaScript is a translated language, here the source code can be visualized, but idea here is still similar to the object’s interface, without considering its implementation process.

Visibility of methods and properties were another idea of information hiding. In few languages, an object has its own public, private or protected methods. The categorization determines the level of access to the users. Public methods can be used by anyone, but only the internal implementation of the object makes use of the private methods. In JavaScript,
entire methods and properties were declared as public, nevertheless there are ways to safeguard the data inside an object and accomplish their privacy.

1.2.4 **Aggregation**

Aggregation has the capability to combine many objects into a new one. Aggregation is the powerful way to classify an issue into smaller manageable parts. It provides a way to think about the issue on various levels of abstraction.

1.2.5 **Inheritance**

Reusing the code which has been written earlier is inheritance. The programmer object requires more-specific functionality for execution, such as methods to write a code which reuses the entire person's functionality.

In OOP, one class inherits from other classes, where as in JavaScript, objects inherit from other objects as there are no classes. It generally includes the methods to the inherited ones and hence it extends the old object. The object which inherits the methods has the capability to pick one or more methods. It can redefine them, it will adjust for their own use. Interface behaves similar to this, the method name will be same whenever a new object’s call and the method behaves differently. So redefining the inherited method is called as overriding.

1.2.6 **Polymorphism**

The capability to call the same method for various objects and everyone answering in their own way is called as polymorphism.
1.3 SOFTWARE QUALITY

The software’s quality is computed by different variables and these variables can be segregated into external and internal quality criteria. The user’s experiences when executing the software in its operational mode is nothing but external quality. The internal quality is a perspective, where they are code-dependent and the end-user can’t see that. External quality is crucial to the user, only Developers make use of the internal quality.

Some quality criteria were considered as objective and it can be computed accordingly. Some quality criteria were considered as subjective, and hence they can capture with more arbitrary measurements.

The internal quality (code characteristics) is responsible of the developers only. The entire external quality aspects (coming from using the software) were crucial to the end-user. Nevertheless, the developers also focus on the performances (speed, space, network usage) and determinism, to make the testing process easier. According to the developers, the requirements are ease-of-use, back-compatibility, security, and power consumption also it is vital to assume the complexity to compute every criterion. Since, there is no simple variable to look at or because the computation process is costly or because it demands a complex infrastructure, so it seems to be a difficult task. For example, assume an objective measurement, which is simple to compute. Power consumption has simple measurement (how many µW the application consumes), but it is difficult to measure. Security is very complex and it is bit costly to compute.

Features: This is the key reason to write the software to provide service. Features represent the software’s result e.g., a numerical result, a
string, a screen shot, a web page, an audio, etc., regardless of the performances (speed, memory).

**Speed:** It explains stronghold of the application to provide the better service. The elapsed time between the user request for the service and the moment, the service is delivered. The sum of the CPU time, system time, and network latency is the considered as the elapsed time or wall time. Hence the developer not only concentrated on the CPU time (the time CPU actually spends on executing the program) also it consider the elapsed time. By the disk usage (a write on the disk is very costly), swapping (due to an excessive virtual memory size), or time spent by the network (latency issue, or too many round trips), the CPU time can easily be overshadowed.

**Space:** Consider the application’s storage space such as the aggregate numbers like peak memory, virtual memory size, etc. The principal effect on the speed of the applications is conceived when the data triggers a CPU cache miss or a disk write. A mediocre data design gives poor performances.

**Network usage:** The bandwidth and latency calculates the network usage. Mismanaging sockets and channels provides insignificant time spent in opening and closing sockets, handshakes, and round trips. As for memory, caching approach will minimize the usage of network resources.

**Stability:** Frequent patching is required to fix the software defects. For the user, this is a difficult process. For the developer, the code is delicate and it might benefit from proper testing or partial rewrite.

**Robustness:** Frequently the application stale, freezes, or crashes. The tolerability of the software in its extreme conditions i.e. restricted
CPU and memory/disk/network resources, corner cases, system failure or unresponsive 3rd-party resources. Based on the testability and coverage this aspect works.

**Ease-of-use:** It can be a very subjective factor and difficult to quantify. It calls for the user documentation, clarity of the error message, exceptions management, and recovery after failure.

**Determinism:** Determinism is also termed as repeatability. The programs generate the same result for the same input and it have many reasons for the program to provide a non-repeatable behaviour. For the user, a non-repeatable behaviour is mystifying and aggravating one. This also makes the program as a complex process to test and debug. Repeatability is strongly dependent on a good data model design.

**Back-compatibility:** It is required by the user, a new version should be cost effective for the current data it is termed as compatibility.

**Security:** It provides access permission of the data. This data can be processed by the application and to be managed. This is a tedious concept of various applications and it is getting more and more tedious to execute the dissemination of mobile and web-based software.

**Power consumption:** Power usage is significant with mobile applications. As a program has to assume how it controls the devices.

**Test coverage:** The percentage of code that is implemented by some unit or regression test. It can calculate this through number of lines, number of functions and number of control branches, which are exploited by tests. General expectation on coverage will be at least 85% for any moderately complex application. So attain this coverage, testability should be improved.
Which in turn, it has deep inference on the architecture and development methodology.

**Testability:** Testability tells how frequently the code development is overlooked or simply ignored aspect? Moreover it has the ability to trigger any particular line of code or branching condition. Highly testable code needs an authorized architecture and development, which is a tedious process to identify. Re-design is required to fix poorly testable software. This demands major investment in software architecture, design, and development methodologies.

**Portability:** This application run on 32 and 64 bits machines, it has the ability to run on a mobile phone, run on multiple OS (e.g., Windows, Linux, Mac OS-X, Solaris, iOS, Android, RIM) as well as it runs smoothly on all web browsers (IE, Firefox, Chrome, Safari, Opera).

**Thread-safeness:** This component always free from threads, this application does not get into a deadlock. The concurrency is still the result of a manual process (there is no compiler that automatically parallelizes the code). These were tedious to assure the good functioning of a program also its performance is not unusual to see the program running slower, when too many threads are available, as the cost of synchronization will be high and it is primary.

**Conciseness:** Conciseness is also termed as compactness. A compact code generally points to the faster compilation and smaller binary size it naturally tends to fewer bugs, since the number of bugs is historically constant with aspect to code size.
Maintainability: It is a significant aspect and this process is also tedious to quantify. Maintainability increases when the testability and flexible (abstract) design is good.

Documentation: This is a subjective topic. Some user requires plain documentation in normal English. Some other users demand at least 30% of the code should be comments. Finally disagree that the code itself is the best documentation such as names of the types, classes, functions and arguments.

Legibility: Legibility is also termed as readability and this is also a subjective topic. It explains how easily user can read the code. Guidelines were provided to unify the style of the code, so that a developer can easily read the code of another developer. Code guidelines abound and they go from a small set of directives to a full set of rules that specify every syntactical aspect of the language.

Scalability: It is easy to lengthen a feature or to include a new one or to add extra cores or maximize the size of the cluster application runs on it. This also depends on the software architecture and anticipating future needs.

Software quality can be acquired from the user experience. But software quality cannot be a reactive action to external defects. Software quality is constructed from the ground up with design, development methodologies, with a special concentration on testability, coverage, and flexibility.
1.4 SOFTWARE METRICS

Software metric is the computation of a single characteristic of program efficiency or performance and also it is utilized to compute the attributes of software products and processes. Currently the software development based on Object-oriented (OO) Paradigm is well-defined. The Object-Oriented paradigm for the software development varies from conventional procedural, so the conventional metrics can’t be enforced on OO software.

A number of OO software metrics have been suggested by researchers and practitioners to calculate the quality of OO software. Table 1.1 gives the basic definitions of software metric.

<table>
<thead>
<tr>
<th>Software Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMC</td>
<td>Summation of the complexities of all class methods</td>
</tr>
<tr>
<td>NOC</td>
<td>Number of immediate sub-classes subordinate to a class in the class hierarchy</td>
</tr>
<tr>
<td>DIT</td>
<td>Maximum height of the class hierarchy</td>
</tr>
<tr>
<td>CBO</td>
<td>Number of other classes to which it is coupled</td>
</tr>
<tr>
<td>RFC</td>
<td>A set of methods that can potentially be executed in response to a message received by an object of that class</td>
</tr>
<tr>
<td>LCOM</td>
<td>Measures the dissimilarity of methods in a class via instanced Variables</td>
</tr>
<tr>
<td>NOM</td>
<td>Number of methods defined in a class</td>
</tr>
<tr>
<td>NOA</td>
<td>Number of attribute defined in a class</td>
</tr>
<tr>
<td>Software Metric</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NOAI</td>
<td>Counts the number of attribute which are inherited by all member subclasses</td>
</tr>
<tr>
<td>NOMI</td>
<td>Counts the number of method which are inherited by all member subclasses</td>
</tr>
<tr>
<td>Fan-In</td>
<td>It defines as the summation of number of local flows into that procedure and the number of data structures from which that procedure retrieves information</td>
</tr>
<tr>
<td>Fan-Out</td>
<td>It defines as the summation of number of local flows out of that procedure and the number of data structures that the procedure updates</td>
</tr>
<tr>
<td>NOPM</td>
<td>Number of private methods in a class</td>
</tr>
<tr>
<td>NOPA</td>
<td>Number of private attribute in a class</td>
</tr>
<tr>
<td>NOPM</td>
<td>Number of public methods in a class</td>
</tr>
<tr>
<td>NOPA</td>
<td>Number of public attribute in a class</td>
</tr>
<tr>
<td>NLOC</td>
<td>It is used to measure the size of a program by counting the number of lines in the text of the source code.</td>
</tr>
</tbody>
</table>

1.5 SOFTWARE FAULTS

OOPS makes use of abstraction and encapsulation concepts. These ideas are different from the other coding languages. Abstraction and encapsulation is nothing but acquiring the properties and the methods of the external objects with the help of an object. JavaScript libraries will create the JavaScript easier. Abstraction is called as hiding the unnecessary details after initiating the process. The objects will hide the internal details like how it works and how it calls with the help of the library files. In order to do the task, call the respective methods for the respective objects and then library files,
which are included in the program will do the rest of the action. The code within the objects in the library is encapsulated. So that is executed in its exact way which in turn it won’t affect the library methods. The library authors can rewrite the codes within the objects in their library. The public interface of the library methods can’t be changed at any cause, so there is no need to modify any code to utilize their modified objects.

Variety of bugs occurs in OOPS concepts, in this research those bugs were recognized and examined. OOPS is a client side scripting language, which has an merit features for generating the well-established web pages by utilizing different kinds of functions and modules. User fulfilment will be minimized through these bugs and also it tends to failure of the web page creation. The desirable feature of OOPS concept is to examine the bugs in the OOPS language and prioritize into proper fault category. So, the fault localization methodology is brought-in, which rectifies the issue by providing a detail description about the fault and fault localisation of the examined faults in the OOPS language. To divide the faults according to their categorization of TSVM (Transductive Support Vector Machines) classification methodology is brought-in. It have variety of fault categories, they are explained further.

- **Undefined/Null Variable Usage**

  Each and every variable in OOPS concept has to be declared before use them. In this case try to utilize the variable without declaring, it leads to bug. Also objects or methods have to be declared otherwise it causes error.

- **Undefined Method**

  The methods should be declared as similar to variables. For instance: try to call a method drink ( ) without it, leads to bug.
• **Incorrect Method Parameter**

The values passed to the function should be in an exact format or type, else it lead a bug. The parameters that are to be invoked in the method required to be defined during the function declaration. If bug occurs, try to send wrong value at the time of function call. For instance: the software implementation will be failed when try to send string values rather than the integer values to the function setDate().

• **Incorrect Return Value**

Faults in the logical programming will return a wrong value during the incorrect return value error occurred situations.

• **Syntax-Based Fault**

Every programming language, generally have some syntax rules, Similar to that OOPS language, which were pre-defined in the OOPS bin files. When the coding is computed without following these rules in the OOPS bin files it results in error. So, this is called Syntax-Based Fault (SBF). For instance: a word defined in a single quotation rather than the double quotation.

• **Range Based Fault**

Every value has a specific range. This type of faults happens when the passing parameters values for the specific attributes resides in values. This differs according to the parameter range that prevails among them.
• **Incorrect object support**

The methods and properties of an object were determined. The objects which are inappropriate to the idea of the methods or functions, this incorrect object support will raise the error.

1.5.1 **Error Localization**

Initially the nature of the fault can be identified, that creates the program execution failure by examining the errors that occurs in the OOPS language. This can be proceeding by two steps.

- Error collection
- Error analysis

These two steps are followed to identify the appropriate where the error occurs. Initially, in the error collection step examining and recognizing the errors in the OOPS coding will be performed. In an iterative manner by implementing the OOPS code with the slight modifications, the errors can be recognized. Once after collecting the errors evidences, that error will be examined to know the nature of the evidences and type of errors has occurred.

1.5.2 **Error Collection**

In the error collection phase, the existence of error will be examined and searched with respect to the code. This is performed in the script by examining the respective errors. The errors can be created during the code creation, which leads to software failure. Hence the code required to be examined line by line during deployment the phase.
The errors in the OOPS coding is collected according to the following criteria:

(1) Identify the function which has the error line
(2) The function to which the respective line number belongs to it
(3) Variable and functions in the error lines
(4) The previous values of the corresponding variables in the previous execution

After gathering these errors, their nature of errors has to be examined and reason for these errors also recognized. This process of the errors that occurs in the OOPS.

1.6 FAULT PREDICTION

If the errors in the software systems are left as it is, it leads to major issues. Likewise, several systems were given to the users with extreme faults. This is regardless of a large amount of development effort minimizes the faults with respect to quality control and testing. An effective approach to fault reduction is to focus much on recognizing the fault prone parts of the system and targeting those parts for maximizing the quality and testing. In the earlier days a limited work is preceded in that area. In spite of this complexity, it is a tedious approach to recognize the fault-prone software components. The target of our software metrics computes the quality of the object oriented software products. Different attributes defines the software quality, which involves the maintainability, defect density, fault proneness, normalized rework, understandability, reusability etc.

Software metrics (method-level, class-level, component-level, file-level, process-level, quantitative values-level) recognizes the software
faults in the software development life cycle before initiating the testing process. Statistics, machine learning, machine learning along with statistical methods and statistical models vs. expert estimation (Catal et al. 2009) were the methods which are utilized here. In order to calculate the fault prone modules for the next release of software the industry utilizes the earlier software metrics and fault data. Software fault prediction approaches required much cost to identify the software faults which is distinguished with the software reviews (Catal et al. 2010).

1.6.1 Statistics Based Approach

The faulty software components were identified in the earlier stage of development and a prediction model is constructed with the help of object oriented classes as soon as possible to detect the faulty classes. The statistics based approaches were broadly utilized. The export coupling metrics were the famous one to recognize the fault prone classes (Emam et al. 2001). The prediction model has a high accuracy but has a low design quality. They gather the data from the commercial java applications and recognize the fault classes for the future release of applications. The design and code metrics (Zhao et al. 1998) will distinguish the accuracy of fault prediction models which are available before and after the system. Code metrics were accessible only after the system is established and design metrics were accessible before initiating the coding process. With the help of linear regression and according to the data from one release the models were established for a huge telecommunication system which was developed by Ericsson. In this work, prediction is done after the system is 34% more accurate when compared before the system. The variability of metrics available before and after the implementation is 43% and 58% (Tomaszewski et al. 2005) but the metrics aren’t utilized the performance of the system will be similar.
Generally, the file has huge number of faults in the next release of system software. To recognize the faults in every file of the next release of a system, a negative binomial regression model is established. This gives the fault’s count for every file of the release according to the following characteristics:

The file size,

- The file was new to release
- File was changed or unchanged from the previous release
- The number of faults in previous release
- The age of the file and the programming language

The average percentage of faults was recognized by the model is 20% and the prediction of faults is accurate (Ostrand et al. 2005), this will happen according to our study. In order to recognize the fault-prone in software modules, the software metrics were accessible early in the software development life cycle. A predictive model is constructed to raise the prediction accuracy with the help of metrics, which explains the textual requirements, available static code metrics and combination of both requirement metrics and static code metrics. Three NASA projects data CM1, JM1 and PC1 (Jiang et al. 2007) were utilized by our researchers. These three requirement metrics helps in defect prediction.

1.6.2 Machine Learning Approach

The software fault prediction approach is the best approach to recognize the faults and also raises the software reliability and quality. Data is classified into clusters with the help of fuzzy subtractive clustering and then
the Radial Basis Function (RBF) network is enforced to estimate the software faults. Accuracy of RBF is better, when compared with the multilayer perception (MLP) but inspection cost and completeness value of MLP’s is greater (Mahaweerawat et al. 2002). In order to give the software quality with the help of median-adjusting class labels, a neural network classifier with metric inputs and class labels were utilized. It makes use of

- Multilayer perception

- Three multilayer analysis

- A test set of metrics with non-adjusted quality class labels

An effected pre-processing method is nothing but Median-adjusted class labels, which estimate a software object quality. Performance of component wise Median is bad because of lack of orthogonal invariance (Pizzi et al. 2002). In object oriented software systems, a classification approach is utilized to identify the faults. The Multilayer Perception (MLP) works on the fault prediction model. Predictions of faults were examined in classification process and it is further classified based on the fault variety. This is according to the clustering and RBF. In this study, MLP is utilized to recognize the faulty classes and RBF Neural network categorize faults, based in many varieties of faults which were determined in faulty classes (Mahaweerawat et al. 2004).

A Support Vector Machine (SVM) based software class model is established for an early software quality prediction when only a small number of sample data are available. In order to divide the software program modules into fault-prone and non fault-prone categories, many techniques were used. For data classification, SVM technique is utilized. SVM is robust in nature when compared with the other techniques for software quality prediction and it
is adjustable to modelling nonlinear functional relationships. An efficient technique for software quality prediction is utilized to accomplish a good performance (Xing et al. 2005). Researchers also enforce a machine learning approach on real-time software systems to compute the software defects. For instance: tele-control/ tele-presence, robotics and mission planning systems (Challagulla et al. 2005). Many prediction techniques (Statistical models such as Stepwise Multi-linear Regression models and multivariate models, and machine learning models such as Artificial Neural Networks, Instance based Reasoning, Bayesian-Belief Networks, Decision Trees and Rule Induction) were utilized, but there is no such technique to provide an accurate result for entire data sets. Size and complexity metrics aren’t enough for predicting real time software defects.

1.6.3 Statistical and Machine Learning Approach

In the Mozilla open source software system, in order to estimate the fault proneness of the code. The Researchers make use of statistical methods (Logistic regression and linear regression) and Machine learning techniques (Decision trees and neural networks). Based on them, performance of Lines of Code (LOC) metric is well and correctness of Lack of Cohesion on Methods (LCOM) metric is perfect, but its unity value is low. Multivariate models performance is good for fine grained analysis (Gyimothy et al. 2005). Researchers estimate the Fault-prone Software modules with the help of Statistical and Machine learning methods. They distinguish the logistic regression with Machine learning methods Artificial Neural Network, Decision Tree, Support Vector Machine (SVM), cascade correlation network, group method of data handling polynomial method) in order to detect the consequence of static code metrics on fault proneness. Researchers utilizes the Receiver Operating Characteristic (ROC) curves for the modules predicting as
fault-prone or non-fault-prone, to define the performance by computing the Area Under Curve (AUC) from a ROC Curve (Singh et al. 2010).

1.6.4 Statistical models vs. Expert estimation

Researchers distinguish the accuracy of statistical prediction models with the expert estimation on two huge telecommunication systems. In order to estimate the Fault-prone of the code units they make use of two approaches, statistical fault prediction model and human experts. Prediction on the class level and component level is also done. Statistical models were inexpensive to construct and they can be utilized without experts and they perform accurately on small and large systems. On huge systems, expert estimation is limited (Kanmani et al. 2007).

1.6.5 Nearest Neighbor Techniques

It have several sampling techniques that were utilized conventionally as like random, stratified, systematic and clustered but all these techniques make use of the class attributes not a non-class attributes. So, they utilize the J48 and Naïve Bayes on five NASA defect datasets. Every dataset is classified into three groups such as a training set, nice neighbour test sets and nasty neighbour test sets. The result shows that accuracy of nice experiments is 94% and nasty experiments are 20% (Boetticher and Gary D 2005). In clustering and classification, the K Nearest Neighbour (KNN) method technique is utilized.
1.7 PROBLEM STATEMENT

The problems that are detected at the time of software development process are given below:

- The software may tend to wrong output, because of erroneous happened in some modules during the execution phase.
- The software can be established wrongly because of fault appeared at the design phase.
- Bug, it occurs in the particular module would affect entire software because of coupling and coherence properties
- It is tedious to estimate and recognize the fault after establishing software because of the occurrence of different modules and inter-linkage functions

1.8 MOTIVATION OF THE RESEARCH

Generating an efficient system is the primary challenge for software developers, who are related with reliability based issue as they construct and execute. The quality of a product is important in the development process of any software product as the quality corresponds to the satisfaction of the customer without concession of the organization’s advantages. The object oriented coding is used in the software product development. The reorganization of the faults in these codes is important in enhancing the quality to construct a novel framework which effectively identifies the faults which occurs in the software accurately.

The previous sections highlighted the basic concepts of Software Process Improvement and its importance, and the role of Medium-Sized software. Software organizations face a host of issues that are different from
those faced by big organizations. These were also elaborate prior. The research attempts to address the challenges faced by software organizations in the domain of Software Process Improvement. In this domain as well, the challenges faced by organizations are distinct and unique.

Software Process Improvement is absolutely vital for any software organization interested in improving its market. This research attempts to address some of these problems and suggest solutions. In this direction, the research first presents an empirical case study of software fault prediction. Measurement is vital for any software process improvement initiative. This research aimed at providing solutions to problems of software process improvement in software organizations. The research also provides some recommendations to improving this situation. The research then attempts to exploit the potential of promising advancements in the software fault prediction. In this direction, the research attempts to using the structural and instructor information of the class. Evaluates predicting the faults based on malicious code present in the system. This is definitely expected to aid the organization in its attempt to software process improvement. The research also applies measuring changeability of object oriented classes and packages using hybrid probabilistic and SVM-ELM model. The estimation of process quality would definitely aid the organization in identifying the faulty processes and thereby contribute to process improvement in future projects.
1.9 OBJECTIVE OF THE RESEARCH

The objective of the research work is to deploy the framework, which identifies the software faults perfectly. Considering the accurate estimation of the software fault by examining the information about the software with the help of machine learning algorithms which assist in examining the patterns, representation of attributes and OpCodes, which occurs in the software. So that faults can be predicted perfectly.

1.10 RESEARCH CONTRIBUTION

The primary part of the proposed research methodologies are listed as follows:

- Fault prediction with the help of structural and instructor information of the class.
- Estimating the faults based on malicious code present in the system.
- Software fault prediction based on changeability of classes and packages.

1.11 ORGANIZATION OF THE THESIS

**Chapter 1:** Focuses on the overview of the problem definition, objectives and contribution of the present research work.

**Chapter 2:** Establishes the different current methodology which are considered as the motivation of the research work.

**Chapter 3:** Explains and evaluates fault prediction by using the structural and instructor information of the class.
Chapter 4: Explains and evaluates predicting the faults based on malicious code present in the system.

Chapter 5: Explains and evaluates measuring changeability of object oriented classes and packages using hybrid probabilistic and SVM-ELM model.

Chapter 6: Provides the performance evaluation discussion of the entire research work in terms of various performance metrics

Chapter 7: Concludes this research work and the scope for future enhancement is also discussed in this chapter.