CHAPTER 1

INTRODUCTION

In the information explosion era of 21st century, software development is considered as one of the most specialized services. The influence of software technologies on scientific and daily life has grown exponentially during the last few decades. From a simple digital clock to telecommunication networks, software is extensively used to ease the way of living. Software is now regarded as the gateway to future, as it provides the basis for much of the technological advance in this century by Card et al (2008). Software is critical in providing a competitive edge to a great number of organizations and is progressively becoming a key component in business systems, products and services. The advancements in information technology, computers and intelligent devices have increased the use of software systems in all aspects of modern society.

Software systems are becoming more and more complex and at the same time, people’s quality expectations are also increasing. Therefore, it is necessary to manage these expectations using an engineering discipline called Software Quality Engineering. Quality assurance, an important task of Software Quality Engineering, is the planned and systematic way of monitoring the methods and processes of a software to ensure quality.

Software Quality Engineering consists of many quality assurance activities such as fault prevention, inspection, fault tolerance, formal verification and fault prediction. These subsets can detect many software problems and can even improve the testing process. Improvements in testing process will reduce development life cycle, project risks, resource and infrastructure costs.
The quality of the software can be measured in terms of various attributes such as fault proneness, maintenance effort and testing effort. Among these attributes, usage of fault proneness to increase software quality is a much sought after problem by both researchers and academicians. Fault proneness is defined as the probability of fault detection in a software class or module ((Malhotra & Jain (2012); Singh et al (2010); Aggarwalet al (2009); (Pai 2007)). The reason behind such popularity is due to the fact that owing to the high complexity and constraints involved in the software development process, it is difficult to develop and produce software without faults. High cost is involved in finding and correcting faults in software projects. Thus, it is necessary to identify or locate the areas where more attention is needed in order to find as many faults as possible within a specified time and budget. In these situations, usage of software fault proneness as quality predictor is desired.

A fault is defined as a problem in software that when executed causes a failure. Fault proneness is the likelihood of a piece of software to have faults. Fault prediction is identified as one major area to predict the probability that the software contains fault. Currently, software quality metrics use defective data to construct prediction model for the next release of software product. Predictive modeling is the process by which a model is created or chosen to try to best predict the probability of an outcome. The objective of a fault-proneness model is to identify faulty classes and focus testing effort on them. Predictive models are built using the software metrics and fault data belonging to previous software for fault prediction. This knowledge is then used to improve the quality of the next release of the software.

Usage of testing tools to identify faults in design and codes and is generally a time consuming process requiring high man power and repeated execution of the test cases. The testing and verification efforts are especially very high for large software systems (Klein 2009). In these situations, it is always desirable to have a prediction method that can identify fault-prone modules or
classes. Later, only these modules are probed during testing thus saving time and cost of software quality maintenance. Faulty modules can be predicted using either a supervised approach (classification) or an unsupervised approach (clustering). Both the techniques rely on information about various errors or bugs identified in earlier period. This research study proposes the use of supervised approach to design and develop prediction model that can detect and identify faulty modules.

To have an accurate prediction method, an adequate set of metrics that can efficiently characterize error prone classes is vital. Several dozens of metrics have already been proposed and used successfully (Fenton 1991). Realizing the importance of predicting faulty modules, several software companies have also built their own product metrics. TRW (Boehm 1988), the Software Engineering Laboratory (SEL) by McGarry et al (1994) and Hewlett Packard (Basili 1995) are some examples of such organization that have built their own product metrics for fault prediction. However, the present day software industries are using Object Oriented technology for software development extensively. Object Oriented analysis/design methods, Object Oriented languages and Object Oriented development environments are currently popular worldwide both, in small and large software organizations.

The usage of Object Oriented technology during software development, however, has created new challenges in the design of fault prediction models. The traditional software metrics are inadequate and are not applicable to Object Oriented paradigm for identifying software quality and predicting faults. A solution to this issue is to use Object Oriented software metrics for predicting faulty classes in Object Oriented applications.

Usage of software metrics to evaluate the quality of software design has attracted software industries as they help to assess large software system quickly at low cost. Several studies have focused on evaluating the usefulness of software
metrics to predict software design faults (Damm & Lundberg (2007); Catal & Diri, (2009)). These techniques can be loosely categorized as statistical techniques, structural patterns based techniques, and software metrics based techniques, formal / relational concept analysis and software inconsistency management techniques. Classification, a frequently used data mining technique, has found wide usage in a range of problem domains such as finance, medicine, engineering, geology and physics. Combining software metrics and classification is a methodology that has gained attention recently. This study proposes a methodology that combines software metrics and a suite of classifiers (ensembling) to design a fault prediction model.

Existing design metrics include traditional simple metrics, program complexity metrics, CK (Chidamber and Kemerer) Metrics and MOOD (Metrics for Object Oriented Design)Metrics all of which have been extensively used in prediction of faulty modules both in general non-object oriented (non-Object Oriented) systems and objected oriented (Object Oriented) systems. The non-Object Oriented metrics have the disadvantage that they have no firm theoretical base for demonstrating normal fault prediction behavior (Weyuk 1988) and do not consider object oriented paradigms like inheritance, encapsulation and passing of message. This makes them unsuitable for Object Oriented systems. Because of these reasons, the traditional metrics are normally combined with Object Oriented metrics while using with Object Oriented systems.

The study proposes enhanced Object Oriented metrics which are combined with traditional simple metrics, specialized Object Oriented-metrics like MOOD and MK metrics. A feature selection algorithm is used to select only those features that are relevant for fault detection during classification. To analyze the performance of these metrics on fault module detection, the study proposes the use of ensemble classifier that uses three frequently used classifiers, Back Propagation Neural Network (BPNN), Support Vector Machine (SVM) and K-Nearest Neighbour (KNN). A novel classifier aggregation method is also
proposed. This chapter provides the introductory materials related to the research topic along with the research objectives.

1.1 SOFTWARE AND SOFTWARE QUALITY

Software quality is a field of study and practice that describes the desirable attributes of software products. It is mainly used to describe and detect software faults. It is defined as the degree to which a system, component or process meets

(i) Specified requirements
(ii) Customer or user needs or expectations

The software industry is putting considerable effort in trying to improve the quality of their products; its main focus has been on software process improvement as an indirect approach to achieve software product quality.

The state-of-the-art technology available today can now provide abundant functionality to satisfy customer’s demand for high quality. Demand for quality is further intensified by the ever increasing dependence of society on software. Billing errors, large-scale disrupted telephone services, etc., are all be traced to the issue of software quality. In this era, quality has been brought to the center of software development process. From the view of software vendors, quality is no longer an advantage factor in the market place; it has become a necessary condition if a company is to compete successfully.

Today, software industry should have two important factors that have an unprecedented impact on software engineering and mission-critical applications

- Software development has to be more efficient
- The quality level of the delivered products has to be high to meet the requirements and to be successful.
Software is defined as a set of computer programs, procedures and possibly associated documentation along with the data pertaining to the operation of a computer system. The 21st century is envisaging a software-intensive life where highly reliable software is an essential system. For example, Therac-25, a computer-driven radiation system, seriously injured and killed patients by massive overdosing (Levenson & Turner(1993)). The business value of a software product results from its quality as perceived by both acquirers and end-users.

Software product quality engineering is a field of study and practice that describes the desirable attributes of software products. The objective of this field of engineering is to achieve the required quality of the product through the definition of quality requirements and their implementation, measurement of appropriate quality attributes and evaluation of the resulting quality. The objective is, in fact, software product quality, which is defined as the degree to which a system, component or process meets a specified customer requirements or expectations.

Software quality is not an advantage but a necessary factor for software industry (Rizvi & Khan (2009)). Software quality is determined by four factors as listed below (Lloyd & Lipow (1997)).

(i) **Error** - An error is conceptual, syntactic or clerical discrepancy which results in one or more faults in the software.

(ii) **Fault** - Software fault specific manifestation of an error. A discrepancy in the software which can impair its ability to function as intended. In general, an error may be the cause for several faults.

(iii) **Failure** - A software failure occurs when a fault in the computer program is evoked by some input data, resulting in the computer program not correctly computing the required function in an exact manner.
(iv) **Defect** - Either a fault or discrepancy between code and documentation that results in mischief in the testing, installation, maintenance or use of software.

Software quality engineering is concerned with two main aspects, the first is to ensure that the software is developed without faults and second conforms to its user and design specifications. It involves three kinds of activities (Sommerville2001) as follows:

- **Quality assurance**: They must establish organizational procedures and standards that lead to high-quality software.
- **Quality planning**: They must select appropriate procedures and standards and tailor them for a specific software project.
- **Quality control**: They must ensure that procedures and standards are followed by the software development team and should involve produces that avoids errors/bugs in the developed software.

Throughout the software development process some quality characteristics of software become essential. In the literature there are many of these characteristics and they are associated with some measurable metrics. Software quality models are established to determine characteristics that affect quality, to form a set of metrics that measure these characteristics, to collect data that will help to evaluate the quality of software. One of the established quality models is the ISO/IEC 9126-1 by Olsina et al (2001) quality model. According this model, there exists two approaches to software quality control, namely, Defect Management Approach (DMA) and Quality Attributes Approach (QAA) (Figure 1.1).

In defect management approach, a software defect is defined as any failures to address the end-user requirements. Examples include missed requirements, misunderstood or misinterpreted requirements, errors in design, functional logic, data relationships, inadequate or wrong test cases for validity.
checking and code errors. DMA approaches are based on counting and managing defects.

QAA to software quality is best exemplified by fixed quality models, such as ISO/IEC 9126. This standard describes a hierarchy of six quality characteristics, each composed of sub-characteristics. They are, Functionality, Reliability, Usability and Reusability, Efficiency, Maintainability and Portability or Availability.

![Software Quality Control Approaches](image)

**Figure 1.1: Software Quality Control Approaches**
According to ISO 9126-1, Functionality is the software’s capability of performing important functions. It has four sub-characteristics.

- Suitability is the software’s capability of beyond being functional also performing appropriate functions.
- Accuracy is the software’s performance of functioning in a right way and with the right output.
- Interoperability of the software indicates how well it communicates with its environment.
- Security is the closeness of the software functions to the unwanted outside users.

Reliability, according to ANSI/IEEE (1990), is the ability of software to perform its required functions under stated conditions for a specified period of time. It has three sub-characteristics:

- Maturity is the length of the time between failures.
- Fault tolerance is the ability of software to continue to its normal operation under the existence of faults in the system.
- Recoverability is the capability of the software to bring itself back to its normal operation after a failure.

Usability is the easiness for the users of software to learn, understand, and use its functions. It has four sub-characteristics:

- Understandability is the degree of which the purpose of the software is clear to the evaluator.
- Learnability is the degree of easiness of learning functions and usage of the software.
- Operability is the degree of easiness of operating the system in right way.
- Attractiveness is the ability of the software product to attract user’s attention.
Efficiency, also referred as performance, is the effective use of system resources, such as time and storage, to fulfill a task.

- Time behavior covers the response and processing time and throughput rates of the software when a specific task is being completed.
- Resource utilization is the usage of the CPU and memory appropriately when the software performs its functions.

Portability refers to how well the software can adapt to changes in its environment.

- Adaptability is the degree of the system satisfying different system constraints and user needs.
- Install ability characterizes the effort needed to port the software to its usage environment.
- Coexistence is the ability of the software to operate in an environment with other software.
- Replace ability is the easiness of exchanging given software within specified environment.

Maintainability, also referred as supportability, is the convenience about finding bugs in the software and fixing them or modifying the software for expanding its features or to adopt it to a new environment. It has four sub characteristics.

- Analyzability is the ability to find the reasons of the failures.
- Changeability is the easiness of doing changes within the software.
- Stability is the scarceness of risk of failures after a change in the software.
- Testability is the verifiability of the new changes in the software.

To satisfy and incorporate any of these factors, the industry has to adapt to the use of measurements to rank the quality of software products and its modules. One important measurement is the quality metrics which are used to predict the
fault proneness of a software module. According to the ISO 9126-1 Quality Model, metrics to be used are not explicitly defined, rather left to the user to be defined.

Quality management and engineering have enjoyed a diversity of applications over the past few years. For example (Kan2006),

- A system of teaching hospitals conservatively estimates $17.8$ million saved on an investment of $2.5$ million in quality management overall five-year period.
- The US Air Force Military Airlift Command improved capacity through quality improvement problem solving during the Gulf War that they avoided having to deploy civilian aircraft (thus avoiding conveniences like suspension of next-day mail)
- The University of Pennsylvania saved more than $60000$ a year from one project focused on reducing mailing cost.

The list of examples benefited by quality management can grow and can cover a range of industries such as telecommunications, healthcare, law, hospitals, government, pharmaceuticals, railways and schools. As the number of success rate of quality management increased, the software industry also slowly started using quality management tools, which initially involved the usage of test tools only. Due to the increase in quality demand from customers and due to the increase in software size and complexity, the software industry is forced to look out for solutions that could perform quality software engineering by predicting fault programs or modules.

Software quality can be determined and guaranteed through the use of software testing tools and quality metrics. As usage of software testing tools is costly and time consuming, software industries are increasingly using quality metrics to measure and predict software quality. This research work analyzes the
usage of software quality metrics to predict fault prone modules in object oriented software products.

1.2. OBJECT ORIENTED CONCEPTS

Software can also be developed using different programming languages. Depending on what kind of language used, the development of the software can be procedural or object-oriented. Procedural code has limited support for modularizing the software into logical components, while the object-oriented code more often is modularized into logical components. The object-oriented paradigm was created in part to ease the understanding of code and in part to help structure the software in a more maintainable way.

The object oriented approach to design and programming, which was introduced in the 1980s, represents a major paradigm shift in software development. It is predicted that this approach will continue to have a major effect in software for many forthcoming years. Different from traditional programming, which separates data and control, Object Oriented programming is based on objects, each of which is a set of defined data and a set of operations (methods) that can be performed on that data.

Like the paradigm of structural design and function decomposition, the Object Oriented approach has become a major cornerstone of software engineering. In the early days of Object Oriented technology deployment (from late 1980s to mid 1990s), much of the Object Oriented literature was concerned with analysis and design methods. There was little information about Object Oriented development processes. Recent years have accepted object-oriented technology and many applications are based on this technology.

When it comes to designing software, the most popular design technique is Object Oriented design Catal & Diri (2008). Object oriented programming
approach is more maintainable than the procedural ones and it introduces the object concept as well as features like inheritance, encapsulation and polymorphism. Object oriented design increases modularity compared to functional design. Modularity increases understandability thus maintaining the code becomes easier.

As the investigations in this study are all done on object-oriented software, the concept behind object-oriented and some important terms are described in this section.

1. **Objects**: Object oriented programming uses objects and their interactions to build software. An object can be simply defined as an entity that has mainly two parts, namely data (state) and functionality (behavior). Data is stored within objects and it is accessed by means of functionality that the object has. By this way, each object in the software has its own data. According to Shalloway & Trott (2002), this definition is told to reflect just the implementation level description of object. At conceptual level an object is a set of responsibilities like functioning properly and knowing its type and state.

2. **Class**: Class is a first level abstraction of set of objects with common responsibilities (Unhelkar 2005). An object is an instance of a class and class defines the type of the object.

3. **Encapsulation**: Encapsulation refers to hiding the implementation details of a class. By using encapsulation, a data item or the working principles within a class cannot be reached directly (Unhelkar 2005). This introduces the differentiation of the user and the developer of the class. For the developer side, encapsulation gives the easiness of changing data or inner principles of the class without modifying the interface, as well as the classes that use this class’s services.

4. **Inheritance**: Inheritance is the way to derive new classes by using the base properties of a pre-defined class and it is a way of defining higher
levels of abstractions. In the concept of inheritance, there are mainly two classes, base class (also called ancestor class) and the derived class. Inheritance provides diversity by which the derived class inherits the properties of the base class while implementing new features.

5. **Polymorphism**: Polymorphism stands for the change in the behavior under the same interface. In other words, polymorphism allows different objects to respond in its own way to the same method.

6. **Coupling**: Coupling is defined as the dependency of a program module on other modules. According to Mitchell & Power (2003), coupling is referred as a count of the number of non-inheritance related couples with other classes. However, according to Chidamberand & Kemerer (1994) inheritance is also treated as a coupling relation.

There is accumulated experience on object oriented design. One concept that has helped this accumulation is the design pattern which provides reusable solutions for common object oriented problems. Using design patterns will result in a better design with properly used object oriented concepts even for a novice developer because they carry the experience of previous developers. In addition to that, each design pattern addresses a problem-solution pair and even if the problem is very complex, what the pattern addresses is simply expressed with the name of the pattern. Thus design patterns constitute a language between designers to express problem solution pairs (http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html).

1.3. **SOFTWARE METRICS**

Software metrics are important in software engineering and is used for measuring the quality of the products produced. It is important factor that is used to determine the improvements of software processes. Software quality metrics are methods that quantitatively determine the extent to which software process, product or project possess a certain quality attribute. They are used to measure
software engineering products (design, source code, etc), processes (analysis, design, coding, testing, etc.) and professionals (efficiency or productivity of an individual designer). Techniques and methods that identify and predict faults using these quality metrics has gained wide acceptance in the past few decades (Catal & Diri (2009)) as they have direct impact on the software product’s time, cost and scope. The output of software metric analysis is used by senior management to

(i) Evaluate the productivity of the software product
(ii) Establish meaningful goals for improvement of the software engineering process.

To establish goals for improvement, the current status of the software development must be understood and hence software metrics have gained importance. The quality of software can be evaluated using different types of software metrics. Although it has always been the major concern in software development, it still lacks the outline of standards which can measure it by Coleman et al (1994).

Different types of metrics are used by the software industries to analyze the performance of various processes. Some examples include management metrics used for market analysis, finance metrics used for investment analysis and performance metrics to analyze the functioning of individual employees. Software quality metrics are the subset of metrics that focus on software performance. Software quality metrics can be used to evaluate the end-product (end product quality metrics) or can be used to evaluate the in-process software product (in-process quality metric). The essence of software quality engineering is to investigate the relationships among in-process metric, project characteristics and end-product quality, and, based on the findings, engineer improvements in quality to both the process and the product.
The high usage of software system poses high quality demand from users, which results in increased software complexity. Fault prediction is a strategy to identify faulty parts of a program, so that the testing process can concentrate only on those regions. This will improve the testing process and indirectly help to reduce development life cycle, project risks, resource and infrastructure costs. Fault prediction models can be either process oriented (development and maintenance) or product oriented (design and usability).

The day-to-day exhaustive workloads of software project leave less time for strategic thinking to improve software quality. Software project managers are concerned with more mundane but equally issues like developing meaningful project estimates, producing higher-quality systems and delivering product on time. In these situations, usage of software metrics to measure the software quality and identify faults is very helpful and much needed practice.

Additionally, the collection of quality metrics enables an organization to “tune” its software process to remove the “vital few” causes of defects that have the greatest impact on software development. At the project and technical levels, software metrics provide immediate benefit. As the software design is completed, most developers would like to obtain answers to the following type of questions, which can be determined only if metrics have been collected and used as a technical guide.

- Which user requirements are most likely to change?
- Which component in this system is most error prone?
- How much testing should be planned for each component?
- How many errors (of specific types) can be expected when testing commences?

Thus, usage of software metrics to quantify software quality through fault identification and prediction is an important concept and is an active research is
in the field of software engineering. The software quality metrics can be grouped as

(i) Product quality metrics  
(ii) Process quality metrics  
(iii) Maintenance quality metrics.

Product quality metrics are related to customers and focus on predicting customer problems and satisfaction. Some examples include size, complexity, design features, performance and quality level. Process metrics are involved in measuring the effectiveness of defect removal and response time of the fix process, while project metrics focus on number of software developers, cost, schedule and productivity. Process metrics, quantify characteristics of the process being used to develop the software and is used as an efficiency measure to quantify fault detection.

Metrics are independent variables and the fault proneness of a module is the dependent variable (Catal & Banu (2007)). As object oriented paradigm has introduced new concepts and features, new metrics like MOOD for measuring inheritance, polymorphism, coupling and data hiding and Chidamber and Kemerer metrics (CK Metrics) for class level code measurement has been developed.

Object oriented-programming and procedure programming benefit more from product metrics and in particular method level metrics. These measure the complexity level of a code model. When the complexity is high, it usually denotes faulty piece. The primary objectives for object-oriented metrics are similar to that of conventional software metrics and are primarily used

- to better understand the quality of the product  
- to assess the effectiveness of the process  
- to improve the quality of work performed at a project level
Each of these objectives is important, but for the software engineer, product quality must be paramount.

1.4. FAULT PREDICTION MODELS

Software fault prediction is one of the quality assurance activities in Software Quality Engineering. Software fault prediction approaches use previous software metrics and fault data to predict the fault-prone modules for the next release of software. If an error is reported during system tests or in field, that module’s fault data is marked as 1, otherwise it is marked as 0. For the prediction modeling, software metrics are used as independent variables and fault data (1 or 0) is used as the dependent variable by Catalet al (2011).

A project manager needs to make sure a project met its timetable and budget plan without loss of quality. In order to help project managers to make a decision, fault prediction models play an important role to allocate software quality assurance resources. Existing research in software fault-prone models focus on predicting faults from these two perspectives:

- **The number of faults or fault density:** This technique predicts the number of faults (or fault density) in a module or a component. Project managers can use these predictions to determine the process of software timetable and resources allocation. These models typically use data from historical versions (or pre-release parts) and predict the faults in the new version (or the new developed parts). For example, the fault data from historical releases can be used to predict faults in updated releases (Illes-Seifert & Paech (2008); Kim et al (2007)).

- **Classification:** Classification predicts which modules (components) contain faults and which modules don’t. The goal of this kind of prediction distinguishes fault free subsystems from faulty subsystems. This allows project managers to focus resources to fix faulty subsystems. Many software fault prediction models are this type of study (Arisholm...
A general framework of the classification model used for software fault prediction is shown in Figure 1.2. The fault prediction process usually includes two consecutive steps: training and prediction. In the training phase, a prediction model is built with previous software metrics (class or method-level metrics) and fault data belonging to each software module. After this phase, this model is used to predict the fault proneness labels of modules that locate in a new software version.

There are two methods to classify fault-prone modules from fault free modules: supervised learning and unsupervised learning. Both of them are used in different situations. When a new system without any previous release is built, for the new developed subsystems (modules, components, or classes), in order to predict fault-prone subsystems, unsupervised learning needs to be adopted. After some subsystems are tested and put into function, these pre-release subsystems can be used as training data to build software fault prediction models to predict
new subsystems. This is the time when supervised learning can be used. The difference between supervised and unsupervised learning is the status of training data’s class; if it is unknown, then the learning is unsupervised, otherwise, the learning is supervised learning.

A learning is called supervised because the method operates under supervision provided with the actual outcome for each of the training examples. Supervised learning requires known fault measurement data (i.e., the number of faults, fault density, or fault-prone or not) for training data. Usually, Fault measurement data from previous versions (Ostrand et al. (2005)), pre-release (Nagappan & Ball (2005a, 2005b)), or similar project (Khoshgoftaar & Seliya (2003)) can act as training data to predict new projects (subsystems). Most of the reported researches in fault prediction are supervised learning including this research.

Predictive model, alternatively referred to as a classifier, maps historical fault data of some project to its modules, and then uses their complexity metrics to predict faults in newly developed modules. It is assumed that if certain types of software modules were likely to fail in the past, they are also likely to do so in the future. However, accurate predictions require a long fault history, which may not exist for the project at hand; in fact, a long fault history is something one would like to avoid altogether. Hence, projects without prior fault history rely on predictive models developed from other, unrelated projects. In these situations, an unsupervised model is used. Unsupervised models use clustering technique, which work with unlabeled data and are used to group similar faults together, so that any future errors can be directed to the most similar group.

The learning result of supervised is easier to judge than unsupervised learning. This probably helps to explain why there are abundant reports on supervised learning in the literature and there are few reports on unsupervised learning. Like most research conducted in fault prediction, a data with all known
classes is divided into training data and testing data: the classes for training data are provided to a machine algorithm, the testing data acts as the validation set and is used to judge the training models. The success rate on test data gives an objective measure of how well the machine learning algorithm performs. When this process is repeated multiple times with randomized divided training and testing sets, it is the standard data mining practice, called cross-validation. Like other research in data mining, randomization, cross-validation, and bootstrapping are often the standard statistical procedures for fault prediction in software engineering.

Irrespective of the model used, the task of predicting faulty modules is considered as a challenging task because of the following reasons:

- During decentralized software development, some companies may not collect fault data for their components as the execution cost of data collection tools may be expensive
- Company may not collect fault data for a version due to the lack of budget
- In some cases, the developers may not report the faults, which make the prediction process even more difficult

In the current scenario, software fault prediction is the most popular research area in these prediction approaches and recently several research centers started new projects on this area. These studies typically produce fault prediction models which allow software engineers to focus development activities on fault-prone code, thereby improving software quality and making better use of resources. The many fault prediction models published are complex and disparate and no up-to-date comprehensive picture of the current state of fault prediction exists.
1.5. OVERVIEW TO CLASSIFICATION PROCESS

Together with software metrics, the most crucial step in the process of fault prediction is classification. The quality of the software product depends greatly on the success of the classifier and has to be designed carefully. The operation of the classification phase can be simplified as being a transform of quantitative input data to qualitative output information. The output of the classifier may either be a discrete selection of one of the predefined classes, or a real-valued vector expressing the likelihood values for the assumption that the pattern originated from the corresponding class.

Classification, also known as pattern recognition, discrimination, supervised learning or prediction, is a task that involves construction of a procedure that maps data into one of several predefined classes (Montejo-Raez 2005). It applies a rule, a boundary or a function to the sample’s attributes, in order to identify the classes. A classifier works to partition the feature space into decision regions that are identified using pre-defined labels. An efficient classifier should be able to differentiate these partitions with precise decision boundaries (borders between decision regions) (Figure 1.3).

Figure 1.3 : Classifier and Decision Boundaries

The efficiency of a classification technique depends on various factors as follows.

(i) Whether learning method is a supervised or unsupervised method
(ii) Type of label output (binary or multiple)
(iii) Whether they are statistical or non-statistical in nature

Examples include Artificial Neural Network, Decision Tree Classifiers, Support Vector Machines, Naïve Bayes Classifiers and Rule-Based Classifiers.

Each technique employs a learning algorithm to identify a model that best fits the relationship between the attribute set and class label of the input data. The model generated by a learning algorithm should satisfy two conditions as follows.

(i) It should fit the input data well
(ii) It should correctly predict the class labels of records it has never seen before.

For that reason, the primary goal of the learning algorithm is to build models with good generalization capability so as to accurately predict the class labels of previously unknown records.

1.5.1. General Approach to Classification

As mentioned earlier, classification, also known as supervised learning or prediction, is a task that involves construction of a procedure that maps data into one of several predefined classes (Montejo-Raez2005). It applies a rule, a boundary or a function to the sample’s attributes, in order to identify the classes. Classification can be applied to databases, text documents, web documents, web based text documents, etc. Classification is considered as a challenging field and contains more scope for research. Some example applications include predicting tumor cells as benign or malignant, classifying credit card transactions as legitimate or fraudulent, classifying secondary structures of protein as alpha-helix, beta-sheet, or random coil, categorizing news stories as finance, weather, entertainment, sports, etc. and grouping e-mails as spam or non-spams. It is considered challenging (Wu2006) because of the following reasons:
• Information overload – The information explosion era is overloaded with information and finding the required information is prohibitively expensive.

• Size and Dimension – The information stored is very high, which in turn, increases the size of the database to be analyzed. Moreover, the databases have very high number of “dimensions” or “features”, which again pose challenges during classification.

A basic classification model is shown in Figure 1.4.

![Figure 1.4 : Classification Model](image)

The input data for a classification task is a collection of features arranged as in row-wise fashion (records). Each record, also known as an instance or example, is characterized by a tuple \((X, y)\) where \(X\) is the attribute set and \(y\) is a special attribute, designated as the class label (also known as category or target attribute).

Classification is the task of learning a target function \(f\) that maps each attribute set \(X\) to one of the predefined class labels \(y\). The target function is also known informally as a classification model. Here, a training set consisting of records whose class labels are known must be provided. The training set is used to build a classification model, which is subsequently applied to the test set, which consists of records with unknown class labels.
1.5.2. Classification and Prediction

Classification is the process of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of training data (i.e., data objects whose class label is known).

The derived model may be represented in various forms, such as classification (IF-THEN) rules, decision trees, mathematical formulae or neural networks. Out of these the use decision tress representation is more popular as they can be easily converted to classification rules.

Classification can be used for predicting the class label of data objects. However, in many applications, users may wish to predict some missing or unavailable data values rather than class labels. This is usually the case when the predicted values are numerical data and is often specifically referred to as prediction. Although prediction may refer to both data value prediction and class label prediction, it is usually confine to data value prediction and thus is distinct from classification. Prediction also encompasses the identification of distribution trends based on the available data.

1.6. MOTIVATION

The main goal of software engineering is to produce better software with high quality. However, control of quality is impossible unless it becomes quantifiable. Thus, fitting the software quality on a measurable basis is an important work on which many studies have been performed but is yet an open ended subject.

Due to high complexity and constraints involved in the software development process, it is difficult to develop and produce software without faults. High cost is involved in finding and correcting faults in software projects.
Thus, there is a need to identify or locate the areas where more attention is needed in order to find as many faults as possible within a specified time and budget.

Nowadays, machine learning (method of programming computers to optimize performance criterion using example data or past experience) is widely used in various domains (i.e., retail companies, financial institutions, bioinformatics, etc.). There are various machine learning methods available. Most of the studies have concentrated on testing the applicability of the existing machine learning algorithms for software prediction. Amongst the various models predicted, researchers always want to determine one of the models to be the best model, which can be used by researchers in further studies to predict the faulty classes. Studies which have enhanced the performance of the existing classifier to improve prediction process are sparse.

To design a successful fault prediction model, it is very important to understand the relationship of object oriented metrics and fault proneness. In other words, the metrics which have significant impact in predicting the faulty classes has to be identified. Then, these significant metrics can be combined into one set to build the multivariate prediction models for predicting fault proneness. Identified metrics will help software practitioners to focus on fault prone classes and ensure a higher quality software product with the available resources. Software researchers may use these metrics in further studies.

Despite the amount of effort software engineers have been putting into developing fault prediction models, software fault prediction still poses great challenges. The research on the applicability of machine learning and statistical techniques in the field of software engineering and software fault prediction is an ongoing process for more than 20 years, and yet it has not reached its matured state. Unfortunately, none of these prediction models have achieved widespread
applicability in the software industry due to a lack of software tools to automate this prediction process.

Usage of software metrics to evaluate the quality of software design has attracted software industries as they help to assess large software system quickly at low cost. Increased usage of object oriented paradigm has been envisaged in recent software products, which has increased the need for new quality metrics to be devised.

Several studies have focused on evaluating the usefulness of software metrics to predict software design faults (Damm & Lundberg (2007)). These techniques can be loosely categorized as statistical techniques, structural patterns based techniques, and metrics based techniques, formal / relational concept analysis and software inconsistency management techniques. Classification, a frequently used data mining technique, has found wide usage in a range of problem domains such as finance, medicine, engineering, geology and physics. Combining software metrics and classification based prediction models is a methodology that has gained attention recently and is the focus of this research work.

Existing metrics for fault module detection include CK metrics and Mood metrics along with traditional general metrics like simple metrics and program complexity measures. Traditional metrics do not consider object oriented paradigms like inheritance, encapsulation and passing of message and therefore do not perform well with fault prediction. The object oriented metrics have been developed specifically to analyze the performance of object oriented system. But, the increase in software complexity and size is increasing the demand for new metrics to identify flaws in the design and code of software system. This demand has necessitated the researchers to focus on adopting new metrics for which established practices have yet to be developed. This research work meets such needs and demands through the development of four metrics for OO design.
1.7. PROBLEM STATEMENT AND OBJECTIVES

The primary objective of the research work is to combine software engineering software metrics and data mining classification concept to predict faulty modules in object oriented software. For this purpose, the following specific objectives were formulated.

- To develop and propose object oriented software quality metrics for measuring class complexities that can be used as a medium to identify design defects.
- To develop and implement classifiers based on Back Propagation Neural Networks (BPNN), K Nearest Neighbor (KNN) and Support Vector Machines (SVM) that uses the proposed and can traditional quality metrics to predict faulty modules in object oriented program.
- To propose 2-classifier and 3-classifier fusion algorithms that respectively combine any of the two and three of the single classifiers for fault prediction.
- To analyze the applicability of dimensionality reduction algorithm to reduce the curse of dimensionality.
- To analyze four partitioning methods that can efficiently partition the input dataset (quality measures) and generate different variants of input dataset that can be used as input to classifiers.
- To propose a hybrid majority vote and weighting scheme as aggregation method during ensemble classification.
- To conduct performance analysis to analyze the effectiveness of the proposed quality metrics, dimensionality reduction algorithm and the effect of fusion classifiers in terms of accuracy and speed of classification.

To achieve the above objectives, the research problem is formulated as follows:
Let

- ‘\( C_L \)’ be a set of having two values \{Faulty, Not-Faulty\} indicating faulty and error free module.
- \( F = \{f_1, f_2, \ldots, f_n\} \) be an n-dimensional feature vector describing the quality measures of the software product. Each component ‘\( f_i \)’ of ‘\( F \)’ expresses an unique quality metric of an module ‘\( m \)’ in the software.

The proposed fault prediction model is defined as a mapping

\[ D : F_n \rightarrow [0, 1]^c \]

i.e., the output \( D(x) \) is a c-dimensional vector whose \( i^{th} \) component denotes the support for the hypothesis that comes from class.

1.8. LAYOUT OF THE THESIS

The underlying objective of this research work is to design and develop an ensemble fault prediction model for object oriented software products using software metrics. The methods and techniques used by these algorithms and the results obtained during experimentation are reported in this dissertation. The dissertation is organized as follows.

Chapter 1, Introduction provided a brief introduction to software quality and metrics machine learning algorithms. The motivation and objectives of the research work were also outlined.

The literature review is a critical look at the existing research that is significant to the work that is carried out. In case of software faults, several researchers have addressed the problem of software fault prediction. A critical look at the various available literatures related to the present research work is given in Chapter 2, Review of Literature. The research work enhances the fault prediction process through the use of traditional and extended metrics designed to suit object oriented software products along with traditional object oriented
metrics. The research methodology used along with the detailed description of the metrics is presented in Chapter 3, Methodology and Quality Metrics.

The study proposes the use of ensemble of classifiers method for fault proneness prediction. In order to enhance the process of prediction, various feature reduction techniques are used. These feature reduction techniques along with the design details of the prediction models are presented in Chapter 4, Design of EoC Based Fault-Proneness Prediction Models.

The proposed prediction models were vigorously tested with various performance metrics and the results of these experiments are tabulated and discussed in Chapter 5, Results and Discussion. This Chapter identifies the winning model most suitable for software fault prediction in Object Oriented software.

The conclusion of the research work is summarized along with future research direction in Chapter 6, Summary and Conclusion. The work of several researchers are quoted and used as evidence to support the concepts explained in this dissertation. All such evidences used are listed in the reference section of the dissertation.

1.9. CHAPTER SUMMARY

This chapter provided the introductory concepts of topics related to software and software faults along with the research objectives. Recent years have seen a rapid increase in the size and complexity of software. It is increasingly becoming important to have a prediction tool that can foretell faulty modules, which will save time and cost to analysts, development and the management. Classification is the field of data mining, which is used to prediction for the past several decades. This research presents a Classification based model for software prediction. Several researchers have proposed methods
for software fault prediction and some of them are presented in the next chapter, Chapter 2, Review of Literature.