CHAPTER 1

INTRODUCTION

1.1 SOFTWARE ENGINEERING (SE)

The dynamics of today’s corporate business applications is characterized by a constant need for integration and restructuring of existing software which may occur due to redefinition of a corporate strategy, a corporate take-over, a conversion of the existing infrastructure from a data-processing model towards a service-oriented model. This continuous modification process will finally result in a situation where several software systems have to collaborate in a way that was never anticipated in their original design. Such large scale software applications are often referred to as legacy applications. A legacy system is an operational system that has been designed, implemented and installed in a radically different environment than imposed by the current Information Technology (IT) infrastructure.

Reliability assessment of upgraded legacy systems is an important problem in IT software infrastructure. Some parts of the code used in the original design of such systems are currently being discontinued. Maintaining a legacy system, therefore, demands upgradation of the software components. Trustworthy reliability assurance after an upgrade with a new software component is needed which combines the evidence about the reliability of the new component with the knowledge about the old system accumulated to date. In these circumstances, Bayesian approach to reliability assessment is invaluable (Parvinder Singh et.al, 2007). Earlier studies have used Bayesian
inference under simplified assumptions. Here the effect of these on the accuracy of predictions is studied and analyzed, some of which are open for future research, of using various data mining techniques inference for practical reliability assessment.

SE is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. It has emerged as a profession and field of study dedicated to creating software that is cheaper, easier to maintain, quicker and of higher quality to build. Since the field is still relatively young compared to its relative fields of engineering, there is still much work and debate around what SE actually is, and if it deserves the term engineering. It has grown organically out of the limitations of viewing software as just programming. Software development is a term sometimes preferred by practitioners in the industry who view SE as too heavy-handed and constrictive to the integrated process of creating software. The Software Development Life Cycle (SDLC) is the sequence of different activities that take place during development. Any software development consists of five phases, namely Requirement analysis, Design, Implementation, Testing and Maintenance (Pressman 2000).

The computer programming language development can be understood by measuring the complexity of computer programs with respect to the size of the programs. Another way of looking at the evolution of programming languages is getting the computer to accomplish more and more complex tasks. Lack of understanding of a program's overall structure and functionality will result in the failure of detecting errors in the program. This can be avoided by using better languages that conversely reduce the number of errors by enabling a better understanding. This will provide what software design has attempted, considering the software at ever greater levels of abstraction. Such inventions as statement, sub-routine, file, class, template,
library, component and more have allowed the arrangement of a program's parts to be specified using abstractions. This is achieved by layers, hierarchies and modules, which provide structure at different granularities, to make the code comprehensible. In addition, improvements in languages have enabled more exact control over the shape and use of data elements in the abstract data type. These data types can be specified to a very fine degree, including how and when they are accessed and even the state of the data before and after they are accessed (Sommerville 2001).

1.2 SOFTWARE QUALITY

The quality of software depends on its characteristics and performance. The software quality is nothing but the conformance to requirements or program specification related to reliability. It is divided into internal and external quality characteristics. External quality characteristics are those parts of a product that face its users, where internal quality characteristics are those that do not. The user satisfaction is more important than anything in determining software quality. The quality is a varying value to each person and inherently subjective as different people will experience the quality of the same software in a different manner. A computer has no concept of well-written source code. However, from a human point of view, source code can be written in a way that has an effect on the effort needed to comprehend its behavior. Many source code programming style guides often aim readability and language-specific conventions at reducing the cost of source code maintenance.

In the context of SE, software quality measures how well software is designed (quality of design) and how well the software conforms to that design quality of conformance. The quality of conformance is concerned with implementation, quality of design measures and how valid the design and
requirements are in creating worthwhile product. With software embedded into many devices today, software failure has caused more than inconvenience (Kapur and Garg 1992).

1.2.1 Software Quality Factors (SQF)

A Software Quality Factor (SQF) is a non-functional requirement for a software program, which is not called up by the customer's contract, but is a desirable requirement, that enhances the quality of the software program. Understandability is possessed by a software product if the purpose of the product is clear. All of the design and user documentation must be clearly written so that they are easily understandable. This is obviously subjective and the user context must be taken into account, i.e. if the software product is to be used by software engineers, it is not required to be understandable for the layman.

A software product possesses the characteristic conciseness to the extent that no excessive information is present. This is important where memory capacity is limited, and it is important to reduce Lines of Code (LOC) to a minimum. It can be improved by replacing repeated functionality by one sub-routine or function, which achieves that functionality. A software product must have the portability to the extent that it can be operated easily and well on multiple computer configurations.

Software should always be consistent to the extent that it contains uniform notation, symbology and terminology within itself. A software product must be maintainable to the extent that it facilitates updating to satisfy new requirements. Thus the software product, which is maintainable, should be well documented and should have spare capacity for memory usage and processor speed. Software should be testable to facilitate the establishment of
acceptance criteria and support evaluation of its performance. Such a characteristic must be built-in during the design phase if the product is to be easily testable to check whether a complex design leads to poor testability (Musa 1999).

The efficiency of a software product fulfills its purpose without waste of resources. This means resources in the sense of memory utilization and processor speed. A software product must have security to the extent that it is able to protect data against unauthorized access and to withstand malicious interference with its operations. Besides presence of appropriate security mechanisms such as authentication, access control and encryption, security also implies reliability in the face of malicious, intelligent and adaptive attackers/hackers.

1.2.2 Measurement of SQF

There are several perspectives within the field on measurement. There are great measures that are valued by professionals. For any software, quantitative measures are mandatory; but at the same time qualitative measures should not be let incurred for, as qualitative measures are as important as quantitative measures. SQF cannot be measured because of their indistinct description. It is necessary to find measures, or metrics, which can be used to quantify them as non-functional requirements. For example, reliability is an SQF, but cannot be evaluated in its own right. However, there are related attributes to reliability, which can indeed be measured. Such attributes are Mean Time To Failure (MTTF), rate of failure occurrence, availability of the system, etc. Similarly, an attribute of portability is the number of target dependent statements in a program.
1.2.3 Software Quality Assurance (SQA)

Software testing is viewed as an important part of the SQA process. In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the SE processes itself to reduce the number of faults that end up in reduced defect rate. An acceptable defect rate is constituted depending on the nature of the software. Though there are close links with SQA, testing departments often exist independently. In recent years the term “grey box testing” has come into common usage. This involves having access to internal data structures and algorithms for the purpose of designing the test cases. Manipulating input data and formatting output data do not qualify as grey-box because the input and output are clearly outside the black-box. This is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for the test. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages, (Aho and Ullman).

1.3 SOFTWARE TESTING

Software testing is a task intended to detect defects in software by contrasting a computer program's expected results with its actual results for a given set of inputs. By contrast, Quality Assurance (QA) is the implementation of policies and procedures intended to prevent defects from occurring in the initial stage (Sommerville 2001). Software testing, when done correctly, can increase overall software quality of conformance by proving that the product complies with its requirements. Software testing is used in association with Verification and Validation (V and V). V and V are done to check whether the software has met its objectives and match all the specifications and requirements of the clients.
1.3.1 Static and Dynamic Testing

There are many approaches available in software testing. Reviews, walkthroughs or inspections are considered as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. The static testing can be often omitted, whereas the dynamic testing takes place when programs are to be used for the first time, which is normally considered as the beginning of the testing stage. This may actually begin before the program is complete, in order to test particular sections of code of modules or discrete functions. For example, spreadsheet programs are, by their very nature, tested to a large extent “on the fly” during the build process. Because of this feature some calculation or text manipulation is shown immediately after each formula is entered.

1.3.2 Testing Process

A common practice of software testing is performed by an independent group of testers after the functionality is developed before it is delivered to the customer. This practice often results in the testing phase being used as project buffer to compensate for project delays, thereby compromising the time devoted to testing. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project is completed. These tests fail initially; as they are expected to. Then as code is written, it passes incrementally larger portions of the test suites. The test suites are continuously updated as new failure conditions and corner cases are discovered and integrated with any regression tests that are developed. Unit tests are maintained along with the rest of the
software source code and generally integrated into the build process with inherently interactive tests being relegated to a partially manual build acceptance process (Whittaker 2000).

Testing can be done on the following levels. Integration testing exposes defects in the interfaces and interaction between integrated components or modules. Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system. System testing tests a completely integrated system to verify that it meets its requirements. System integration testing verifies that a system is integrated to any external or third party systems defined in the system requirements. Before shipping the final version of software, alpha and beta testing are often done additionally. Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing. Beta testing comes after alpha testing. Versions of the software, known as beta versions, are released to a limited audience outside the programming team. The software is released to groups of people so that further testing can ensure that the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users. Finally, acceptance testing can be conducted by the end-user, customer, or client to validate whether or not to accept the product. Acceptance testing may be performed as part of the hand-off process between any two phases of development.
1.4 SOFTWARE RELIABILITY

Software reliability is an important aspect of software quality. According to American National Standard Institute (ANSI), it is defined as the probability of failure-free operation of a computer program in a specified environment for a specified time. One of reliability's distinguishing characteristics is that it is objective, measurable, and can be estimated, whereas much of software quality is subjective criteria. This distinction is especially important in the discipline of SQA, (Malaiya and Srimani 1990). These measured criteria are typically called software metrics. Although software reliability is defined as a probabilistic function and comes with the notion of time, it must be noted that software reliability is different from traditional hardware reliability and not a direct function of time. Electronic and mechanical parts may become old and wear out with time and usage, but software will not rust or wear-out during its life cycle (Jelinski and Moranda 1972). Software will not change over time unless intentionally changed or upgraded (Yashvanth Nachimuthu 1991).

Software reliability is an important attribute of software quality along with functionality, usability, performance, serviceability, capability, maintainability and documentation. Software reliability is hard to achieve, because the complexity of software tends to be high. While any system with a high degree of complexity, including software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer with the rapid growth of system size and ease of doing so by upgrading the software. While the complexity of software is inversely related to software reliability, it is directly related to other important factors in software quality especially functionality, capability, etc. Emphasizing these features will tend to add more complexity to the software, (Lyu 1995).
1.5 ERROR CLASSIFICATION METHODS

Often the severity of a software defect can vary even though the software never changes. The reason is that a software defect’s severity depends on the system in which it runs. Another problem which occurs regularly is that the definitions of the severity levels or categories themselves change depending on the type of system. Therefore, the system itself determines the severity of a defect based on the context for which the defect applies. The context makes all the difference in how to classify a defect’s severity. Errors, which are agreed as valid, will be categorized as follows: category A, B and C. In category A, the errors are serious that prevent system test of a particular function continuing or serious data type error, category B where serious or missing data related errors will not prevent implementation, and category C where minor errors do not prevent or hinder functionality, (Rock et al. 2003). A type A bug is an either a showstopper or of such importance as to radically affect the functionality of the system. Bugs would be classified as B where a less important element of functionality is affected and C type bugs are mainly cosmetic bugs. Errors are classified into five categories: catastrophic, severe, major, minor and no effect. The defects or errors identified in any software come under anyone of the above categories.

1.6 NAÏVE BAYES BAYESIAN (NB) MODEL

Naïve Bayes Bayes (NB) is one of the most efficient and effective inductive learning algorithms for machine learning and data mining. Its competitive performance in classification is surprising, because the conditional independence assumption, on which it is based, is rarely true in real world applications. Classification is a fundamental issue in machine learning and data mining, (Mitchell 2005) describes the classification of data using NB algorithm. In classification, the goal of a learning algorithm is to
construct a classifier given a set of training examples with class labels. Here the relationship between supervised learning, or function approximation problems and Bayesian reasoning are considered reasoning. The research is started by considering how to design learning algorithms based on Bayes rule. NB models have been widely used for clustering and classification. However, they are seldom used for general probabilistic learning and inference, (Chotirat et al. 2003).

For a wide range of benchmark dataset, NB models learned using accuracy and learning time are comparable to Bayesian networks with context-specific independence (Achcar et al. 1991). Most significantly, NB inference orders the magnitude faster than Bayesian network inference using Gibbs sampling and belief propagation. This makes NB models a very attractive alternative to Bayesian networks for general probability estimation, particularly in large or real-time domains. Mitchell (1997) worked with machine Learning and revealed the various facts in data classification by adopting NB algorithm.
1.7 **NAÏVE BAYES BAYES (NB) CLASSIFIER**

An NB classifier is a simple probabilistic classifier based on applying Bayes theorem with strong (Naïve Bayes) independence assumptions. A more descriptive term for the underlying probability model would be independent feature model. In mathematics, a classifier is a mapping from a discrete or continuous feature space $X$ to a discrete set of labels (Mingxi Wu and Christopher 1999). Classifiers have practical applications in many branches of science and society. Depending on the precise nature of the probability model, NB classifiers can be trained very efficiently in a supervised learning setting. In many practical applications (Yoshimasa Tsuruoka and Junichi Tsujii 1999), parameter estimation for NB models uses the method of maximum likelihood; in other words, one can work with the Naïve Bayes NB model without believing in Bayesian probability or using any Bayesian methods.

Supervised learning is a machine learning technique for creating a function from training data. Maximum Likelihood Estimation (MLE) is a popular statistical method used to make inferences about parameters of the underlying probability distribution from a given dataset. Bayesian probability is an interpretation of probability suggested by Bayesian theory, which holds that the concept of probability can be defined as the degree to which a person believes a proposition. In simple terms, an NB classifier assumes that the presence or absence of a particular feature of a class is unrelated to the presence or absence of any other feature, (Chotirat et al. 2003).

In spite of their naïve design and apparently over-simplified assumptions, NB classifiers often work much better in many complex real-world situations than one might expect. Recently, careful analysis of the Bayesian classification problem has shown that there are some theoretical reasons for the apparently unreasonable efficacy of NB classifiers. An
advantage of the NB classifier is that it requires a small amount of training data to estimate the parameters means and variances of the variables necessary for classification. The advantage of Bayesian models is that various important but non-measurable factors such as software complexity, architecture, quality of V and V activities, and test coverage are easily incorporated in the model. The data then can be classified by the attempts made (Mingxi Wu and Christopher 1999).

1.8 K NEAREST NEIGHBOR

This method is labor intensive when given large training sets, and did not gain popularity until the 1960’s when increased computing power became available. It has since been widely used in the area of pattern recognition. Nearest-neighbor classifier is based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all of the training tuples are stored in an n-dimensional pattern space. When given an unknown, a K nearest neighbor classifier searches the pattern space for the K training tuples that are closed to the unknown tuple. These K training tuples are the K nearest neighbors of the unknown tuples (Kennedy and Eberhart 2001).

Loseness is defined in terms of a distance metric such as Euclidean distance. The Euclidean distance between two points or tuples, say $X_1=(x_{11},x_{12},...,x_{1n})$ and $X_2=(x_{21},x_{22},...,x_{2n})$ is

$$Dist(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2} \quad (1.1)$$
In other words, for each numeric attribute, the difference between the corresponding values of that attribute in tuple \(X_1\) and in tuple \(X_2\) are taken, square this distance, and accumulate it. The square root is taken of the total accumulated distance count. The values of each attribute are then normalized. This helps prevent attributes with initially large ranges such as income from outweighing attributes with initially smaller ranges. Min-max normalization, for example, can be used to transform values ‘\(V\)’ of a numeric attribute \(A\) to ‘\(V\)’ in the range [0,1] by computing

\[
V' = \frac{V - \text{Min}_A}{\text{Max}_A - \text{Min}_A}
\]  

(1.2)

where \(\text{Min}_A\) and \(\text{Max}_A\) are the minimum and maximum values of attributes, \(A\) is the \(K\) nearest neighbor classification; the unknown tuple is assigned the most common class among its \(K\) nearest neighbors. When \(K=1\), the unknown tuple is assigned the class of the training tuple that is closest to it in pattern space. Nearest neighbor classifier can also be used for prediction, that is, to return a real-valued prediction for a given unknown tuple. In this case, the classifier returns the average value of the real-valued labels associated with the \(K\) nearest neighbor of the unknown tuple.

For categorical attributes, a simple method is to compare the corresponding values of the attributes in tuple \(X_1\) with that in the tuple \(X_2\). If the two are identical (e.g., tuples \(X_1\) and \(X_2\) both have the color blue), then the difference between the two is taken as 0. If the two are different (e.g., tuple \(X_1\) is blue but tuple \(X_2\) is red), then the difference is considered to be 1. Other methods may incorporate more sophisticated schemes for differential grading (e.g., where a large difference score is assigned, say, for blue and white than for blue and black).
In general, if the value of a given attribute A is missing in tuple $X_1$ and/or in tuple $X_2$, the maximum possible difference is assumed. Suppose that each of the attribute has been mapped to the range $[0, 1]$ for categorical attributes, the difference values to be 1 are taken if either one or both tuples $X_1$ and $X_2$, then the difference is also 1. If only one value is missing and other is present and normalized, then the difference is 0.

When $K=1$, a test set is used to estimate the error rate of the classifier. This process can be repeated each time by incrementing $K$ to allow for one more neighbor. The $K$ value that gives the minimum error rate may be selected. In general, the larger the number of training tuples is, the larger the value of $K$ will be so that classification and prediction decision can be based on a larger portion of the stored tuples. As the number of training tuples approaches infinity and $K=1$, the error rate can be no worse than twice the Bayes error rate. If $K$ also approaches infinity, the error rate approaches the Bayes error rate. Nearest neighbor classifier uses distance-based comparisons that intrinsically assign equal weight to each attribute. They therefore can suffer from poor accuracy when given noisy or irrelevant attributes. The method, however, has been modified to incorporate attribute weighting and the pruning of noisy data tuples.

Nearest neighbor classifier can be extremely slow when classifying test tuples. If $d$ is a training database of models $D$ tuples and $K=1$, then $O(d)$ comparisons are required in order to classify a given test tuple. By presorting and arranging the stored tuples into search trees, the number of comparisons can be reduced to $O(\log d)$. Parallel implementation can reduce the running time to a constant that is $O(1)$, which is independent of models $D$. Other techniques to speed up classification time include the use of partial distance calculations and editing the stored tuples. In the partial distance method, the distance is computed based on a subset of the n
attributes. If this distance exceeds a threshold, then further computation for the given stored tuple is halted, and the process moves on to the next stored tuple. The editing method removes training tuples that prove useless. This method is also referred to as pruning or condensing because it reduces the total number of tuples stored.

1.9 ARTIFICIAL IMMUNE RECOGNITION SYSTEM (AIRS) – AN OVERVIEW

The Immune System (IS) defends the body against harmful diseases and infections. It is capable of recognizing virtually any foreign cell or molecule and eliminating it from the body. To do this, it must perform pattern recognition tasks to distinguish molecules and cells of the body called “self” from foreign ones called “non-self”. Thus, the problem that the IS faces is that of distinguishing self from dangerous non-self. The number of foreign molecules that the IS can recognize is unknown, but it has been estimated to be greater than $10^{16}$ in the work by Inman (1978). These foreign proteins that are nothing but kinds of molecules must be distinguished from an estimated different proteins of self, so recognition must be highly specific. These are staggering numbers, especially when one considers that the human genome, which encodes the “program” for constructing the IS, only contains about $10^5$ genes.

The architecture of the IS is multilayered, with defenses provided at many levels. The outermost layer, the skin, is the first barrier to infection. A second barrier physiological, where conditions such as pH and temperature provide inappropriate living conditions for some foreign organisms (pathogens). Once pathogens have entered the body, they are handled by the innate IS and by the adaptive immune response. The innate IS consists primarily of circulating scavenger cells such as macrophages that ingest extra
cellular molecules and materials, clearing the system of both debris and pathogens. The adaptive immune response, also called the acquired immune response, is the most sophisticated and involves many different types of cells and molecules (Andrew Watkins and John Timmis 2003). It is called adaptive because it is responsible for immunity that is adaptively acquired during the lifetime of the organism. The adaptive IS provides the most potential from a computer security viewpoint.

Catal and Diri (2007) worked in software detection using AIRS. The work of Xin Jin et al. (2006) attempted to provide software reliability for Software Engineering Management (SEM) and they have taken a list of metrics and implemented in a common dataset. They tried to improve all the measures made by them by incorporating additional metrics with a combined effort. Yi Liu et al. (2007) worked on the problem associated with Software Quality Classification (SQC) modeling and showed that the historical metric dataset obtained from a single software project are often not adequate to build robust and accurate models. To address this issue, multiple dataset obtained from different software projects are used for SQC modeling in recent research works. The previous study has demonstrated that using multiple dataset for validation can achieve robust genetic programming-based SQC models.

Seeker et al. (2004) investigated the effectiveness of using multiple validation dataset. Moreover, a novel general purpose-based classifier, consisting of training, multiple-dataset validation, and voting phases, is proposed. The experiments are carried out on seven National Aeronautics and Space Administration (NASA) software projects. The results are compared with the results achieved by seventeen other data mining techniques. The comparisons demonstrate that the performance of our approach is
significantly better by using multiple dataset from different software projects with similar reliability goals.

According to Timmis et al. (2000), the adaptive IS can be viewed as a distributed detection system which consists primarily of white blood cells, called lymphocytes. Lymphocytes function as small independent detectors that circulate through the body in the blood and lymph systems. Lymphocytes can be viewed as negative detectors, because they detect non-self patterns, and ignore self patterns. Detection, or recognition, of non-self occurs when molecular bonds are formed between a pathogen and receptors that cover the surface of the lymphocyte. The more complementary the molecular shape and electrostatic surface charge between pathogen and lymphocyte receptor, the stronger the bond (or the higher the affinity). Detection is approximate; hence, a lymphocyte will bind with several different kinds of structurally related pathogens.

The ability to detect most pathogens requires a huge diversity of lymphocyte receptors. This diversity is partly achieved by generating lymphocyte receptors through a genetic process that introduces a huge amount of randomness. Generating receptors randomly could result in lymphocytes that detect self instead of non-self, which would then likely cause autoimmune problems in which the IS attacks the body. Autoimmune disorders are rare because lymphocytes are self-tolerant, i.e., they do not recognize self. Tolerance of self is achieved through a process called clonal deletion: lymphocytes mature in an organ called the thymus through which most self proteins circulate; if they bind to these self proteins while maturing, they are eliminated.

Even if receptors are randomly generated, there are not enough lymphocytes in the body to provide a complete coverage of the space of all pathogen patterns; one estimate is that there are $10^{18}$ different lymphocyte
receptors in the body at any given time discussed in Tonegawa (1983). Somatic generation of antibody diversity in nature which must detect potentially $10^{16}$ different foreign patterns. The IS has several mechanisms for addressing this problem, mechanisms which make the immune response more dynamic and more specific. Protection is made dynamic by the continual circulation of lymphocytes through the body, and by a continual turnover of the lymphocyte population. Lymphocytes are typically short-lived (a few days) and are continually replaced by new lymphocytes with new randomly generated receptors. Dynamic protection increases the coverage provided by the IS over time: the longer a pathogen is present in the body, the more likely it is to be detected because it will encounter a greater diversity of lymphocytes.

Protection is made more specific by learning and memory. If the IS detects a pathogen that it has not encountered before, it undergoes a primary response, during which it “learns” the structure of the specific pathogen, i.e. it evolves a set of lymphocytes with high affinity for that pathogen, through a process called affinity maturation. This is a Darwinian process of variation and selection resembling the genetic algorithm. Holland (1992) discusses the high-affinity lymphocytes those that bind most tightly with available pathogens are stimulated to reproduce in great numbers, and the resulting lymphocytes have a large number of mutations. These new mutated lymphocytes then compete for pathogens with their parents and with other clones. Affinity maturation produces a large number of lymphocytes that have high affinity for a particular pathogen, which accelerates its detection and elimination. Speed of response is important in the IS because most pathogens are replicating and will cause increasing damage as their numbers increase. Speed of response to previously encountered pathogens is generally high, because the information encoded in adapted lymphocytes is retained as immune memory. On subsequent encounters with the same antigen pattern the
IS mounts a secondary response. In this case, the adapted lymphocytes eliminate the pathogens so rapidly that the symptoms of the infection are not noticeable by the individual.

Even with all of these mechanisms, the coverage provided by the IS is necessarily incomplete. The consequence is an IS that is vulnerable to particular pathogens. However, not all individuals will be vulnerable to the same pathogens to the same degree, because each individual has a unique immune system. This diversity of IS across a population greatly enhances the survival of the population as a whole. One way in which IS differs from one individual to the next is by having different lymphocyte populations, and hence, different detector sets. Another key component that gives an IS and its uniqueness is the variation in a molecule called Major-Histocompatibility Complex (MHC). MHC molecules enable the immune system to detect intracellular pathogens (e.g., viruses) that reside inside cells. Intra-cellular pathogens are problematic because the inside of a cell is not “visible” to lymphocytes, that is, lymphocytes can only bind to structures on the surface of cells. MHC molecules bind to protein fragments called peptides which could be viral within a cell and transport the peptides to the surface, effectively displaying the contents of the cell to passing lymphocytes. The set of proteins to which an MHC molecule can bind is dependent on the structure of the MHC, which is genetically determined. Each person has only a limited number of MHC types and so is vulnerable to particular pathogens that cannot be readily transported by the available MHC types. However, as a whole, a population is far less vulnerable, because each individual has a different set of MHC types, and so is vulnerable to different pathogens.

To summarize, the natural IS has many features that are desirable from the standpoint of computer science. The system is massively parallel and its functioning is truly distributed. Individual components are disposable and unreliable, yet the system as a whole is robust. Previously encountered
infections are detected and eliminated quickly, while novel intrusions are detected on a slower time scale, using a variety of adaptive mechanisms. The system is autonomous, controlling its own behavior both at the detector and effectors levels. Each IS detects infections in slightly different ways, so pathogens that are able to evade the defenses of one IS cannot necessarily evade those of every other IS.

1.9.1 Artificial Immune System (AIS)

Artificial Immune Systems (AIS) can be defined as abstract or metaphorical computational systems developed using ideas, theories, and components, extracted from the IS. Most AIS aim at solving complex computational or engineering problems, such as pattern recognition, elimination, and optimization. This is a crucial distinction between AIS and theoretical immune system models (Burke et.al 2003). While the former is devoted primarily to computing, the latter is focused on the modeling of the IS in order to understand its behavior, so that contributions can be made to the biological sciences. It is not exclusive, however, the use of one approach into the other and, indeed, theoretical models of the IS have contributed to the development of AIS by De Castro and Timmis (2003).

1.9.2 Biological and Artificial Immune Systems

All living organisms are capable of presenting some type of defense against foreign attack. The evolution of species that resulted in the emergence of the vertebrates also led to the evolution of the immune system of this species. The vertebrate IS is particularly interesting due to its several computational capabilities. The IS of vertebrates is composed of a great variety of molecules, cells, and organs spread all over the
body. There is no central organ controlling the functioning of the IS, and there are several elements in transit and in different compartments performing complementary roles (Yidan Su and Xinyi Gu Zhujuan Li 2006). The main task of the IS is to survey the organism in the search for malfunctioning cells from their own body (e.g., cancer and tumor cells), and foreign disease causing elements (e.g., viruses and bacteria). Every element that can be recognized by the IS is called an antigen (Ag). The cells that originally belong to our body and are harmless to its functioning are termed self or self antigens, while the disease causing elements are named non-self or non-self antigens. The IS, thus, has to be capable of distinguishing between what is self from what is non-self; a process called self/non-self discrimination, and performed basically through pattern recognition events.

From a pattern recognition perspective, the most appealing characteristic of the IS is the presence of receptor molecules, on the surface of immune cells, capable of recognizing an almost limitless range of antigenic patterns. One can identify two major groups of immune cells, known as B-cells and T-cells. These two types of cells are rather similar, but differ in relation to how they recognize antigens and by their functional roles. B-cells are capable of recognizing antigens free in solution (e.g., in the blood stream), while T-cells require antigens to be presented by other accessory cells.

The antigens are covered with molecules, named epitopes. These allow them to be recognized by the receptor molecules on the surface of B-cells, called antibodies (Ab). For an antigen (Ag) to be recognized by a T-cell receptor, it has to be processed and presented by an accessory cell.
Antigenic recognition is the first pre-requisite for the IS to be activated and to mount an immune response (De Castro and Timmis 2002a). The recognition has to satisfy some criteria. First, the cell receptor a or b recognizes an antigen with a certain affinity, and a binding between the receptor and the antigen occurs with strength proportional to this affinity. If the affinity is greater than a given threshold, named affinity threshold, then the IS is activated. The nature of antigen, type of recognizing cell, and the recognition site also influence the outcome of an encounter between an antigen and a cell receptor, shown in Figure 1.1(a).

From Figure 1.1 (b), the human IS contains an organ called thymus that is located behind the breastbone, which performs a crucial role in the maturation of T-cells. After T-cells are generated, they migrate into the thymus where they mature. During this maturation, all T-cells that
recognize self-antigens are excluded from the population of T-cells; a process termed negative selection. If a B-cell encounters a non-self antigen with a sufficient affinity, it proliferates and differentiates into memory and effector cells; a process named clonal selection. In contrast, if a B-cell recognizes a self-antigen, it might result in suppression, as proposed by the immune network theory, shown in Figure 1.1(b).

1.9.3 Negative Selection

The thymus is responsible for the maturation of T-cells; and is protected by a blood barrier capable of efficiently excluding non-self antigens from the thymic environment. Thus, most elements found within the thymus are representative of self instead of non-self. As an outcome, the T-cells containing receptors capable of recognizing these self antigens presented in the thymus are eliminated from the repertoire of T-cells through a process named negative selection by (Nossal 1994), through negative selection of lymphocytes. All T-cells that leave the thymus to circulate throughout the body are said to be tolerant to self, i.e., they do not respond to self.

From an information processing perspective, negative selection presents an alternative paradigm to perform pattern recognition by storing information about the complement set (non-self) of the patterns to be self recognized. A negative selection algorithm by Forrest et al. (1993) has been proposed in the literature with applications focused on the problem of anomaly detection such as computer and network intrusion detection, time series prediction, image inspection and segmentation, and hardware fault tolerance.

Given an appropriate problem representation, define the set of patterns to be protected and call it the self set (P). Based upon the
negative selection algorithm, generate a set of detectors (M) that will be responsible to identify all elements that do not belong to the self-set, i.e., the non-self elements.

(a) Generating the set of detectors  
(b) Monitoring for the presence of undesired (non-self) patterns

Figure 1.2 Pattern recognition via the negative selection algorithm

For all elements of the detector set that correspond to the non-self patterns, check if it recognizes (matches) an element of \( P^* \) and, if yes, then a non-self pattern is recognized and an action has to be taken.

The negative selection algorithm referred in Figure 1.2(a) is executed as follows:

1. Generation of random candidate elements (C) using the same representation adopted;
2. Comparison and matching the elements in C with the elements in P. If a match occurs, i.e., if an element of P is recognized by an element of C, then discard this element of C; else store this element of C in the detector set M. After generating the set of detectors (M), the next stage of the algorithm consists of monitoring the system for the presence of non-self patterns in Figure 1.2(b). In this case, assume a set P* of patterns to be protected. This set might be composed of the set P plus other new patterns, or it can be a completely novel set.

1.9.4 Clonal Selection

Complementary to the role of negative selection, clonal selection is the theory used to explain how an immune response is mounted when a non-self antigenic pattern is recognized by a B-cell, (Ada and Nossal 1987). Figure 1.3 illustrates the clonal selection, expansion (proliferation), and affinity maturation processes. In brief, when a B-cell receptor recognizes a non-self antigen with a certain affinity, it is selected to proliferate and produce antibodies in high volumes. The antibodies are soluble forms of the B-cell receptors that are released from the B-cell surface to cope with the invading non-self antigen (Burnet 1959). Antibodies bind to antigens leading to their eventual elimination by other immune cells. Proliferation in the case of immune cells is asexual, a mitotic process; the cells divide themselves (there is no crossover). During reproduction, the B-cell progenies (clones) undergo a hyper mutation process that, together with a strong selective pressure, result in B-cells with antigenic receptors presenting higher affinities with the selective antigen. This whole process of mutation and selection is known as the maturation of the immune response and is analogous to the
natural selection of species. In addition to differentiating into antibody producing cells, the activated B-cells with high antigenic affinities are selected to become memory cells with long life spans. These memory cells are preeminent in future responses to this same antigenic pattern, or a similar one.

Figure 1.3 Clonal selection, expansion (proliferation), affinity maturation, and maintenance of memory cells

The highest affinity cells are selected to proliferate. Their progenies (clones) suffer mutation with high rates and those whose receptors present high affinity with the antigen are maintained as memory cells.

Forrest et al. (1994) have argued that a Genetic Algorithm (GA) without crossover is a reasonable model of clonal selection. However, the standard GA does not account for important properties such as affinity proportional reproduction and mutation. Some of the referenced authors
De Castro et al. (2000b) proposed a Clonal Selection Algorithm, named CLONALG, to fulfill these basic processes involved in clonal selection. This algorithm was initially proposed to perform pattern recognition and then adapted to solve multi-modal optimization tasks. Given a set of patterns to be recognized (P), the basic steps of the CLONALG algorithm are as follows:

1. Randomly initialize a population of individuals (M);
2. For each pattern of P, present it to the population M and determine its affinity (match) with each element of the population M;
3. Select n1 of the best affinity elements of M and generate copies of these individuals proportionally to their affinity with the antigen. The higher the affinity, the higher the number of copies, and vice-versa;
4. Mutate all these copies with a rate proportional to their affinity with the input pattern: the higher the affinity, the smaller the mutation rate, and vice-versa.
5. Add these mutated individuals to the population M and re-select n2 of these maturated (optimised) individuals to be kept as memories of the system;
6. Repeat Steps 2 to 5 until a certain criterion is met, such as a minimum pattern recognition or classification error.

Note that this algorithm allows the AIS to become increasingly better at its task of recognizing patterns (antigens). Thus, based upon an evolutionary like behaviour, CLONALG learns to recognize patterns.
1.9.5 Immune Network

The immune network theory proposes that the IS has a dynamic behavior even in the absence of external stimuli by Jerne (1974). It is suggested that the immune cells and molecules are capable of recognizing each other, which endows the system with an Eigen-behaviour that is not dependent on foreign stimulation. Several immunologists have refuted this theory, e.g. Langman and Cohn (1986). However, its computational aspects are relevant and it has proved itself to be a powerful model for computational systems.

![Figure 1.4 Immune network theory](image)

The recognition of antigen by an antibody (cell receptor) leads to network activation, while the recognition of an idiotope by another antibody results in network suppression. Antibody Ab2 is said to be the internal image of the antigen Ag, because Ab1 is capable of recognizing the antigen and also Ab2 (Timmis 2000).

According to the immune network theory, the receptor molecules contained in the surface of the immune cells present markers,
named idiotopes, which can be recognized by receptors on other immune cells. These idiotopes are displayed in and/or around the same portions of the receptors that recognize non-self antigens. Figure 1.4 provides a simple illustration of the immune network theory.

To explain the network theory, assume that a receptor antibody Ab1 on a B-cell recognizes a non-self antigen (Ag). Assume now that this same receptor Ab1 also recognizes an idiotope i2 on another B-cell receptor Ab2. Keeping track of the fact that i2 is part of Ab2, Ab1 is capable of recognizing both Ag and Ab2. Thus, Ab2 is said to be the internal image of Ag, more precisely, i2 is the internal image of Ag. The recognition of idiotopes on a cell receptor by other cell receptors, lead to ever increasing sets of connected cell receptors and molecules. Note that the network in this case is a network of affinities, which differ from the ‘hardwired’ network of the nervous system. As a result of the network recognition events, it was suggested that the recognition of a cell receptor by another cell receptor results in network suppression, while the recognition of an antigen by a cell receptor results in network activation and cell proliferation. The original theory did not account explicitly for the results of network activation and/or suppression, and the various artificial immune networks found in the literature immune filter model is a particular form.

Recently, the most influential artificial immune network models found in the literature are De Castro and Von Zuben (2001) and Timmis (2000). Due to limited space, we will restrict ourselves to the description of only one of these two network models, for an overview of Timmis (2000) and AIS with reference to Timmis and Neal (2001). The work presented in De Castro and Von Zuben (2001) makes use of CLONALG, described to explain how the immune network model responds to non-self antigens i.e., becomes activated. The recognition of cell receptors by
other cell receptors results in network suppression. This is modeled by eliminating all but one of the self-recognizing cells. Given a set of patterns (P) to be recognized, the basic algorithm runs as follows:

1. Randomly initialize the network population
2. For each antigenic pattern in P apply the CLONALG algorithm that will return a set of memory cells (M*) and their co-ordinates for the current antigen
3. Determine the affinity (degree of matching) among all the individuals of M*
4. Eliminate all but one of the individuals in M* whose affinities are greater than a given threshold. The purpose of this process is to eliminate redundancy in the network by suppressing self-recognizing elements
5. Concatenate the remaining individuals of the previous step with the remaining individuals found for each antigenic pattern presented. This will result in a large population of memory individuals M
6. Determine the affinity of the whole population M and suppress all but one of the self-recognizing elements. This will result in a reduced final population of memory cells that recognize and follow the spatial distribution of the antigens.
7. Repeat Steps 2 to 6 until a pre-defined stopping criterion is met such as a minimum pattern recognition or classification error.
1.10 PROBLEM STATEMENT

Numerous works have considered the classification of dataset using Naïve Bayes (Parvinder Singh et.al, 2007) and K nearest neighbor algorithm without pre-processing of data to remove imbalance and noise. It is proposed to investigate Naïve Bayes classification and the K nearest neighbor classification on the KC3 Dataset. It is also proposed to generate data pre-processing algorithms using immune techniques found in AIS and uses the K nearest neighbor for classification and study the improvement in classification. Work is also carried out in areas of normalizing data using density-based algorithms, using Neural network techniques to improve the reliability of classification. As newer technique in data processing along with reliability algorithm (SRBIT) are applied, the classification methodology is improved.

1.11 ORGANIZATION OF THE THESIS

Based on the objectives stated, the work has been carried out and presented as given below:

Chapter two describes the Software metrics classification by using Naïve Bayesian model and K nearest neighbor model to ensure the software reliability.

Chapter three presents the proposed Software Reliability Based on Immune Techniques (SRBIT) algorithm and Immune Filter with the KC3 dataset to create reliable dataset for software quality.

Chapter four explains the classification methods based on neural network with data normalization using normal density methods.
Chapter five covers the analysis and comparison between the error report generated using K nearest neighbor and the error report generated using K nearest neighbor with immune filter along with neural network methods.

Chapter six discusses the conclusion and the scope for further research.