Chapter 2

Review of Related Literature

2.1 Introduction

The objective of this chapter is to show the related literature pertaining to the study. The purpose of the literature review is to give the reader the historical development and the state-of-the-art theoretical background on which this thesis is based. A closer look is taken at those metrics still in use or still being cited and those too young to be discarded or cited. In the present study, sound representative literatures are reviewed. The reviews cover the six broad subject domains involved with the present research problem i.e. Basic Object-Oriented Concepts, Object-Oriented Development Life Cycle, Software Complexity, Complexity Metrics in the Object-Oriented Paradigm, Complexity Metrics and Reliability, and Metrics Validation and Measurement Theory.

The presentation of the reviews maintains a very traditional form arranged by the year of availability. The reviews are presented the literature covering the six broad categories/facets however, all the literature reviewed are not listed under different sections or facets enumerated. The reason being that individual literature found fit for this review did not show distinct characteristics pertaining to a particular topic. Sometimes an individual literature covers three to four facets, even a single book covers all the facets enumerated, so to avoid this problem the present organization was preferred. This is maintained just to minimize the vast literature collected, reviewed and to have bibliographical control. All kinds of literature were included i.e., books, journal articles, Ph.D and Master’s thesis, reviews, reports and arranged in ascending order so that the development of the area could be followed. Some of the previous studies relevant to the present research problem that deserve mention are reviewed as under:

Grady and Caswell (1987) defined software metrics in a standard way of measuring some attribute of the software development process. For example, size, cost, defects, communications, difficulty, and environment are all attributes. Examples of attributes in the physical world are mass, length, time and the like. "Kilo-lines of executable
source" illustrates one metric for the size attribute. Metrics can be primitive (directly measurable or countable, such as counting lines of code) or computed (such as non-comment source statements/ engineer/ month. The present work was benefitted with this literature to get the foundational glimpse for the present work.

Weyuker (1988) has developed a formal list of nine properties for software metrics and has evaluated a number of existing software metrics using these properties. These properties include notions of monotonicity, interaction, non-coarseness, non-uniqueness, and permutation. This formal list of properties is uniform for traditional and object-oriented approach but Weyuker's 9th property has received a mixed response regarding its applicability to object-oriented software metrics even through these properties are most frequently used criteria for analytical evaluation of design metrics. In this research the design metrics are analytically evaluated using these properties. The scholar viewed this as a fundamental resource to conceptualize the models.

Li & Henry (1993) state that metrics for the object-oriented paradigm have yet to be studied. Since terminology varies among object oriented programming languages, the authors considered the basic components of the paradigm as objects, classes, attributes, inheritance, method, and message passing. They proposed five metrics; Data Abstraction Coupling (DAC), Number of methods (NOM), Message Passing Coupling (MPC), and Number of semicolons per class (Size1), Number of methods per attributes (Size2). Thus this research concentrates on several object oriented software metrics and the validation of these metrics with maintenance effort in two commercial systems.

Chen & Lu (1993) in their paper presented a new metric for the object-oriented design. The metric measures the complexity of a class in an object-oriented design. The metrics include operation complexity, operation argument complexity; attribute complexity, operation coupling, class coupling, cohesion, class hierarchy and reuse. An experiment is conducted to build the metric system. The approach is to derive a regression model of the metrics based on the experimental data. Moreover, the subjective judgment by the expert is incorporated in the regression model. This ensures that the metric system is pragmatic and flexible for the software industry.
Chidamber & Kemerer (1994) in their paper proposed a metrics suite for object-oriented design, as need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. The theoretical base chosen for the metrics was the ontology of Bunge. Six design metrics Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between object classes (CBO), Response For a Class (UFC), Lack of Cohesion in Methods (LCOM) are developed, and then analytically evaluated against Weyuker’s proposed set of measurement principles. An automated data collection tool was then developed and implemented to collect an empirical sample of these metrics at two field sites in order to demonstrate their feasibility and suggest ways in which managers may use these metrics for process improvement. In the present work this paper is used as a base model.

Lorenz & Kidd (1994) in their book, ‘Object-Oriented Software Metrics’ provide a number of specific metrics that apply to object-oriented software projects. The metrics are based on measurements and derived advice drawn from a number of actual projects that have successfully used object technology to deliver products. Presents a novel metrics-based approach for detecting design problems in object-oriented software. It also introduced an important suite of detection strategies for the identification of different well-known design flaws as well as some rarely mentioned ones. This was used as a fundamental book in the work.

Belin, Tyagi & Tyler (1994) in the study entitled “Object-oriented metrics: an overview” found that Object-Oriented Analysis and Design (OOAD) techniques appear to be at the forefront of software engineering technologies. Nevertheless, as with the introduction of any relatively new technique, there is a tendency for people to attempt to maximize efficiency without always having a corresponding factual basis for their actions. This paper discussed important "bullets of measure" that should be taken into consideration during and after the development of an Object-Oriented System, particularly as it pertains to the static analysis of OO source code. The proposed metrics are consistent with the suggestions of many individuals who are well known for their experience, so it was important for our study.
Abounader (1995) questioned about metrics how these are not properly validated nor extensively tested. Many of these metrics were created and labeled as "design" metrics, which suggests their application at the design level of the software life cycle. In practice, however, most of the so-called design metrics are applied either to code, or to programming language-dependent design notations. This research is based on the premise that design metrics should better be applied to design documents, independently of the implementation or the programming language and that metrics which cannot be derived from such design documents should not be classified as design metrics. They surveyed proposed object-oriented metrics. This literature especially helps the scholar in the present study by giving idea about the information collected at design level before the coding starts.

Sellers & Sellers (1995) in this book the authors offer theoretical and empirical tips and facts for creating an OO complexity metrics (measurement) program, based on a review of existing research from the last several years. Object-oriented (OO) metrics are an integral part of object technology - at the research level and in commercial software development projects. It covers moving through object-oriented concepts as they are related to managing the project lifecycle; the framework in which metrics exist; OO product metrics; and current industrial applications as a knowledge domain for software developers, programmers, and managers.

Cant & Henderson (1995) presented the approach to complexity metrics which is based on an understanding of the cognitive processes of the analyst or programmer as they undertake the challenges of program development or maintenance. In their complexity model they attempted to quantify a number of cognitive processes involving the software developer or maintainer. The authors claim that this cognitive complexity model is applicable to object-oriented systems too from their initial studies. This study provoke the scholar to take cognitive theory as theoretical base for association of design metrics and fault induced at class level.

Neal (1996) in this study looks at each of fifty proposed object-oriented metrics and scrutinizes them for validity Questions raised like does the metric measure what its author proposes to measure? Are the statistics used with the metric appropriate,
considering the scale attributed to the metric? Does the metric hold up under vigorous scrutiny of the conditions of representation and uniqueness? Do intuitive and empirical understandings survive under all allowable transformations? Fifty (50) proposed object-oriented software metrics are scrutinized. Because of duplication of metrics among authors, forty-four (44) different metrics are identified. Nine measures are found to be valid ratio scales through the extensive structure. Five measures are found to be valid ratio' scales through set union concatenation. Five averages are found to be valid ratio scales. Twenty metrics are found to be incapable of validation by measurement theory on all scales: four because of the use of subjective tables, three because of mixed attributes nine because they are non-discriminate, two do not fit the weak order, and two averages. This was very much helpful for understanding various measures in the study.

Briand, Emam & Morasca (1996) these authors of this famous ‘Briand et al’ model contributed substantially to the world of software complexity metrics studies. Elements of measurement theory have recently been introduced into the software engineering discipline. It has been suggested that these elements should serve as the basis for developing, reasoning about, and applying measures. For example, it has been suggested that software complexity measures should be additive, that measures fall into a number of distinct types (i.e., levels of measurement: nominal, ordinal, interval, and ratio), that certain statistical techniques are not appropriate for certain types of measures (e.g., parametric statistics for less-than-interval measures), and that certain transformations are not permissible for certain types of measures (e.g., non-linear transformations for interval measures). In this paper we argue that, in spite of the importance of measurement theory, and in the context of software engineering, many of these prescriptions and proscriptions are either premature or, if strictly applied, would represent a substantial hindrance to the progress of empirical research in software engineering. This argument is based partially on studies that have been conducted by behavioral scientists and by statisticians over the last five decades. The scholar was immensely benefited by this model in analytically analysing the complexity metrics.

Basili, Briand & Melo (1996) in their work analyzed eight medium sized projects in C++ using logistic regression to enhance the practical utility of object-oriented
metrics and found that some CK-metrics Weighted Methods per Class (WMC), Coupling between Class (CBO), and Response for a Class (RFC), Depth of Inheritance (DIT), and Number of Children (NOC) were related to fault-proneness of classes. The RFC and WMC were found significant when object-oriented metrics were analyzed using logistic regression method on three industrial applications developed in C++. The literature was highly cited in the present work as well considered as one of the basic model for the research.

Mustafa & Khan (1997) in their book entitled “Software testing: concept and practices” present the testing concepts and their importance. Authors highlight how the bugs can cause airplane crashes, allowed space shuttle missions to fail, halted trading on the stock market etc. So debugging is a narrow view of software testing, which can be performed to find out design defects. This book is useful in our study to find the knowledge about the errors, faults and their detection during the testing phase of development.

Chidamber Darcy & Kemerer (1998) asserted that with the increasing use of object-oriented methods in new software development there is a growing need to both document and improve current practice in object-oriented design and development. In response to this need, a number of researchers have developed various metrics for object-oriented systems as proposed aids to the management of these systems. In this research an analysis of a set of metrics proposed by (Chidamber and Kemerer) is performed in order to assess their usefulness for practicing managers. First, an informal introduction to the metrics is provided by way of an extended example of their managerial use. Second, exploratory analyses of empirical data relating the metrics to productivity, rework effort, and design effort on three commercial object-oriented systems are provided. The empirical results suggest that the metrics provide significant explanatory power for variations in these economic variables, over and above that provided by traditional measures, such as size in lines of code, and after controlling for the effects of individual developers. The scholar gained important things from this study as employment of previous studies as well as more description about object-oriented design.
Zongtian, Biao & Dahong (2000) in their ten years old study stated that, software metrics has a long history for more than forty years, but the research on object-oriented (OO) software metrics has been going on for a few years only. C&K metrics is one of the most famous researches on OO software metrics. First, this paper analyses the shortcoming of the C&K metrics suite for object-oriented design and provides an improved metrics suite. Then the paper introduces a practical C++ measurement tool, SMTCPP, implemented by the authors based on improved metrics. SMTCPP parses C++ programs by the LL(1) method, extracts a lot of program information, such as classes, members and objects; counts the indications, such as the number of methods per class, the biggest complexity among methods, depth of inheritance tree, the number of children, coupling between object classes, response for class, and relative lack of cohesion in methods. The measure values are very useful to guide the software process. The tool may also put the values into a database to collect sufficient data for building a software quality evaluation model. Last, the paper analyses the experiments for three practical programs. The result shows that SMTCPP is useful. The work was benefitted by this how popular CK model was criticised and the limitations of the said model.

Rumbaugh, Blaha, Premerlani & Lorensen (2001) in this multi-authored book seems to be a popular among the students as a textbook, the authors emphasized that object-oriented technology is more than just a way of programming. Often, books on this subject rely too heavily on programming and coding, thus forcing the readers to think in terms of the machine and not the application. This book applies object technology to provide a practical and productive way to develop software development. It presents a new object-oriented software development methodology – from analysis, through design, to implementation. It provides the conceptual clarity to form theoretical basis of this research.

Emam, Benlarbi, Goel & Rai (2001) in their study logistic regression was used to analyze the data of a large telecommunication application having 174 C++ classes, and found WMC, RFC and CBO metrics effective in predicting the fault-proneness of classes, but became insignificant after controlling the size of the system.
Li (2002) is of the opinion that in complex software systems, reliability is the most important aspect of software quality, a multi-dimensional property including other factors like functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software reliability engineering is becoming a standard, widespread practice applicable to the different phases of the software development process. The first chapter of the survey provides an introduction to software measurement. Traditional and object-oriented software metrics are analyzed in detail, comprehensive study of some empirical work is also provided in order to validate the usefulness of the selected software metrics. An overview of software reliability is then introduced from the basic terminologies to the reasons for the need of software reliability. Following is the classification of the existing software reliability measures and measurement tools discussed in several chapters. Firstly, the procedure of software reliability measurement procedure along with a framework is addressed. Secondly, software reliability modeling is introduced in detail together with model classification schemes. Thirdly, the relationship between software reliability engineering and Software development process is outlined. Fourthly, we show a classification of current development tools with some usage information. In the final part of this survey, the research directions of software reliability engineering are explored. This helped the scholar in shaping the fundamental concepts of the area.

Yu, Systa, & Muller (2002) conducted a validation study of CK metrics with data from the client side application of a large network service management system, which contained 123 Java classes. The dependent variable once again was the fault-proneness of the classes is predicted using the linear regression and discriminant analysis. It was found that most CK metrics (except DIT) were sound predictor for the fault-prone classes.

Bansiya & Davis (2002) in their paper describes an improved hierarchical model for the assessment of high-level design quality attributes in object-oriented designs. In this model, structural and behavioral design properties of classes, objects, and their relationships are evaluated using a suite of object-oriented design metrics. This model relates design properties such as encapsulation, modularity, coupling, and cohesion to
high-level quality attributes such as reusability, flexibility, and complexity using empirical and anecdotal information. The relationship, or links, from design properties to quality attributes are weighted in accordance with their influence and importance. The model is validated by using empirical and expert opinion to compare with the model results on several large commercial object-oriented systems. A key attribute of the model is that it can be easily modified to include different relationships and weights, thus providing a practical quality assessment tool adaptable to a variety of demands.

**Vinter, Loomes, & Kornbrot (2002)** in their historic model-like paper gifted a solution to the world of software metrics. It is generally accepted that failure to reason correctly during the early stages of software development causes developers to make incorrect decisions which can lead to the introduction of faults or anomalies in systems. Most key development decisions are usually made at the early system specification stage of a software project and developers do not receive feedback on their accuracy until near its completion. Software metrics are generally aimed at the coding or testing stages of development, however, when the repercussions of erroneous work have already been incurred. This paper presents a tentative model for predicting those parts of formal specifications which are most likely to admit erroneous inferences, in order that potential sources of human error may be reduced. The empirical data populating the model was generated during a series of cognitive experiments aimed at identifying linguistic properties of the Z notation which are prone to admit non-logical reasoning errors and biases in trained users. Like many other researchers the scholar too made the benefit of this model.

**Kan (2003)** in his definitive book with second edition asserts that it covered the essential topics of software development. Comprehensive in scope with extensive industry examples, it shows how to measure software quality and use measurement to improve the software development process. Four major categories of quality metrics and models are addressed: quality management, software reliability and projection, complexity, and customer view. In addition, the book discussed the fundamentals of measurement theory, specific quality metrics and tools, and methods for applying metrics to the software development process.
Bandi, Vaishnavi, & Turk (2003) explained the Object-Oriented (OO) paradigm has become increasingly popular in recent years. Researchers agree that, although maintenance may turn out to be easier for OO systems, it is unlikely that the maintenance burden will completely disappear. One approach to controlling software maintenance costs is the utilization of software metrics during the development phase, to help identify potential problem areas. Many new metrics have been proposed for OO systems, but only a few of them have been validated. The purpose of this research is to empirically explore the validation of four existing OO design complexity metrics and, specifically, to assess their ability to predict maintenance time. This research reports the results of validating four metrics, Interaction Level (IL), Interface Size (IS), Operation Argument Complexity (OAC), and Attribute Complexity. A controlled experiment was conducted to investigate the effect of design complexity (as measured by the above metrics) on maintenance time. Each of the four metrics by itself was found to be useful in the experiment in predicting maintenance performance. It helps the scholar in providing the background about the research design and research methodology.

Purao, Sandeep, Vaishnavi & Vijay (2003) surveyed the metrics proposed for object-oriented systems, focusing on product metrics. The survey is intended for the purposes of understanding, classifying, and analyzing ongoing research in object-oriented metrics. The survey applies fundamental measurement theory to artifacts created by development activities. We develop a mathematical formalism that captures this perspective clearly, giving appropriate attention to the peculiarities of the object-oriented system development process. Consistent representation of the available metrics, following this mathematical formalism, shows that current research in this area contains varying coverage of different products and their properties at different development stages. The consistent representation also facilitates several analyses including aggregation across metrics, usage across metrics, equivalent formulation of metrics by multiple researchers, and exploitation of traditional metrics for object-oriented metrics. We also trace the chronological development of research in this area, and uncover gaps that suggest opportunities for future research.
Succi, Pedrycz, Stefanovic & Miller (2003) in their research paper used data from two applications implemented using C++. One had 150 classes and 23 KSLOC (thousands of SLOC), while the other 144 classes and 25 KSLOC to evaluate the influence of six CK metrics on the number of faults. They found out that RFC and DIT were the most valuable predictors.

Subramanyan & Krisnan (2003) analyzed the data collected from a large B2C e-commerce system, which was developed in C++ and Java to work on AIX and Windows NT. They only examined metric data of WMC, CBO and DIT from 405 C++ classes and 301 Java classes. They applied linear regression with Box-Cox transformation to determine the relationships between metrics and number of faults. It was found that the impact of the metrics was not consistent between two programming languages, although size was always an effective indicator.

Chae & Kwon (2004) examined the existing metrics and presents an approach for improving the cohesion metrics by considering the characteristics of the dependent instance variables in an object-oriented environment. The study shows that it is not possible to evaluate the correct cohesiveness of the class without considering the dependent instance variables.

Pandian (2004) in his book analyzed the fundamentals of software measurement, including scales, levels and instruments. It explained how to design and apply metric system architecture and also examined the methods of data visualization and analysis through empirical data. It explored metrics for defect management including defect measurement and classification. This fundamental book helped the scholar in shaping some basics of software metrics and their analysis.

Kanmani, Uthariaraj , Sankaranarayanan & Thambidurai (2004) in their paper exposed multi-authored discussed the results arrived at employing Object-Oriented (OO) measures on the small-sized programs developed by the Under Graduate (UG) students during the study of C++ laboratory course. The metric values computed reflect the experience/knowledge of the developer in various mechanisms (inheritance, coupling and cohesion) in developing the modules (classes). We propose
six hypotheses to validate the measures. For this experiment, the number of attributes and number of methods defined in the class are correlated with the metric values. The result of the experiment shows that the programs used inheritance and cohesion properties appropriately in the design of the class level attributes.

Nagappan & Ball (2005) the authors were of the opinion that during software development it is helpful to obtain early estimates of the defect density of software components. Such estimates identify fault-prone areas of code requiring further testing. They present an empirical approach for the early prediction of pre-release defect density based on the defects found using static analysis tools. The defects identified by two different static analysis tools are used to fit and predict the actual pre-release defect density for Windows Server 2003. Further, the predicted pre-release defect density as a reliability measure and the actual pre-release defect density are strongly correlated at a high degree of statistical significance showing that the reliability can be estimated during the testing phase. Discriminate analysis shows that the results of static analysis tools can be used to separate high and low quality components with an overall classification rate of 82.91%. This study gives clear cut identification of pre released and post release failure data in establishing the concept of empirical validation of design metrics.

Sarker (2005) explained in her masters thesis that the object oriented design is becoming more popular in software development environment and object oriented design metrics is an essential part of software environment. This study focus on a set of object oriented metrics that can be used to measure the quality of an object oriented design. The metrics for object oriented design focus on measurements that are applied to the class and design characteristics. These measurements permit designers to access the software early in process, making changes that will reduce complexity and improve the continuing capability of the design and reduce the overall efforts of development. This report summarizes the existing metrics, which will guide the designers to support their design. They have categorized metrics and discussed in such a way that novice designers can apply metrics in their design as needed. This study is helpful for the scholar in understating the concept of object design and their measurement.
Counsell, Swift & Tuckern (2006) described an empirical investigation using object-oriented (OO) classes as a basis. Twenty-four respondents (drawn from IT experienced and novice groups) were asked to rate ten classes sampled from two industrial systems in terms of their overall cohesiveness; a class environment was used to carry out the study. Three hypotheses were investigated as part of the study, relating to class size, the role of comment lines and the differences between the two groups in terms of how they rated cohesion. Several key results were observed. Firstly, class size (when expressed in terms of number of methods) only influenced the perception of cohesion by novice subjects. Secondly, well-commented classes were rated more highly amongst IT experienced than novice subjects. Thirdly, results suggest strongly that cohesion comprises a combination of various class factors including low coupling, small numbers of attributes and well-commented methods, rather than any single, individual class feature per se. Finally, if the research supports the view that cohesion is a subjective concept reflecting a cognitive combination of class features then cohesion is also a surrogate for class comprehension. This helps the scholar in understanding the use of cohesion in class level design to implementation.

Counsell, Swift & Crumpton (2006) propagated the concept of cohesion in a class has been the subject of various recent empirical studies and has been measured using many different metrics. In the structured programming paradigm, the software engineering community has adopted an informal yet meaningful and understandable definition of cohesion based on the work of Yourdon and Constantine. The object-oriented (OO) paradigm has formalised various cohesion measures, but the argument over the most meaningful of those metrics continues to be debated. Yet achieving highly cohesive software is fundamental to its comprehension and thus its maintainability. In this article we subject two object-oriented cohesion metrics, CAMC and NHD, to a rigorous mathematical analysis in order to better understand and interpret them. This analysis enables them to offer substantial arguments for preferring the NHD metric to CAMC as a measure of cohesion. Furthermore, they provide a complete understanding of the behaviour of these metrics, enabling us to attach a meaning to the values calculated by the CAMC and NHD metrics. In
addition, they introduced a variant of the NHD metric and demonstrate that it has several advantages over CAMC and NHD. While it may be true that a generally accepted formal and informal definition of cohesion continues to elude the OO software engineering community, there seems considerable value is being able to compare, contrast, and interpret metrics which attempt to measure the same features of software.

Zhou & Leung (2006) found that in the last decade, empirical studies on object-oriented design metrics have shown some of them to be useful for predicting the fault-proneness of classes in object-oriented software systems. This research did not, however, distinguish among faults according to the severity of impact. It would be valuable to know how object-oriented design metrics and class fault-proneness are related when fault severity is taken into account. In this paper, they use logistic regression and machine learning methods to empirically investigate the usefulness of object-oriented design metrics, specifically, a subset of the Chidamber and Kemerer suite, in predicting fault-proneness when taking fault severity into account. Their results, based on a public domain NASA data set, indicate that 1) most of these design metrics are statistically related to fault-proneness of classes across fault severity, and 2) the prediction capabilities of the investigated metrics greatly depend on the severity of faults. More specifically, these design metrics are able to predict low severity faults in fault-prone classes better than high severity faults in fault-prone classes.

Aggarwal, Singh, Kaur & Malhotra (2006a) stated that increasing importance of software measurement has led to development of new software measures. Many metrics have been proposed related to various constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. In this paper they investigate 22 metrics proposed by various researchers. The metrics are first defined and then explained using practical applications. They are applied on standard projects on the basis of which descriptive statistics, principal component analysis and correlation analysis is presented. Finally, a review of the empirical study concerning chosen metrics and subset of these measures that provide sufficient information is given and metrics providing overlapping information are excluded from the set.
Aggarwal, Singh, Kaur & Malhotra (2006b) clarified that the importance of software measurement is increasing leading to development of new measurement techniques. As the development of object-oriented software is rising, more and more metrics are being defined for object-oriented languages. Many metrics have been proposed related to various object-oriented constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. The applicability of metrics developed by previous researchers is mostly limited to requirement, design and implementation phase. Exception handling is a desirable feature of software that leads to robust design and must be measured. This research addresses this need and introduces a new set of design metrics for object-oriented code. Two metrics are developed that measure the amount of robustness included in the code. The metrics are analytically evaluated against Weyuker’s proposed set of nine axioms. These set of metrics are calculated and analyzed for standard projects and accordingly ways in which project managers can utilize these metrics are suggested.

Mustafa & Khan (2007) in their book focused on software testing in practice and it has been planned to suit the needs of both the practitioners and the academicians. Concepts of software testing have been modeled as phase-embedded activity rather than treating them as separate and post-development activity. Each chapter starts with a set of objectives, with the prospective of targeting to achieving rather than leaving the student directionless and ends with a list of key terms, referring to certain abstract concepts for better and crisp communication along with a list of references to enable the user to find in-depth information.

Breesam (2007) viewed that object oriented design is becoming more popular in software development environment and object-oriented design metrics are essential parts of software environment. Many new metrics are being proposed for object-oriented systems, but only few have been validated. The primary purpose of this paper is to analytically and empirically validation of set of metrics that can be used to measure the quality of an object-oriented design in terms of the using class inheritance (generalization and specialization). Analytical description for each metric is considered depending on the obtained results from the practical use of these. This was helpful in generalizing the view of object-orient inheritance metrics.
Nagappan, Williams, Vouk, & Osborne (2007) studied the industrial practice and claimed that the information on the software field quality of a product is available too late in the software lifecycle to guide affordable corrective action. An important step towards remediation of this problem lies in the ability to provide an early estimation of post-release field quality. This paper evaluates the Software Testing and Reliability Early Warning for Java (STREW-J) metric suite leveraging the software testing effort to predict post-release field quality early in the software development phases. The metric suite is applicable for software products implemented in Java for which an extensive suite of automated unit test cases are incrementally created as development proceeds. We validated the prediction model using the STREW-J metrics via a two-phase case study approach which involved 27 medium-sized open source projects, and five industrial projects. The error in estimation and the sensitivity of the predictions indicate the STREW-J metric suite can be used effectively to predict post-release software field quality.

Aggarwal, Singh, Kaur & Malhotra (2007) explained that demand for quality software has undergone rapid growth during the last few years. This is leading to an increase in the development of metrics for measuring the properties of software such as coupling, cohesion or inheritance that can be used in early quality assessments. Quality models that explore the relationship between these properties and quality attributes such as fault proneness, maintainability, effort or productivity are needed to use these metrics effectively. The goal of this work is to empirically explore the relationship between object-oriented design metrics and fault proneness of object-oriented system classes. The study used data collected from Java applications containing 136 classes. They use a set of twenty-six design metrics in work. Result of this study shows that many metrics are based on comparable ideas and provide redundant information. It is shown that by using a subset of metrics in the prediction models can be built to identify the faulty classes. The proposed model predicts faulty classes with more than 80% accuracy.

Pai & Dugan (2007) presented a methodology for Bayesian analysis of software quality and discussed the aspect of relating internal metrics to external quality metrics. They built a Bayesian network (BN) model to relate object-oriented software metrics to software fault content and fault proneness. Assuming that relationship can
be described as a generalized linear model and derived paramedic functional forms for the target node conditional distribution in the BN. The modules are empirically evaluated using a public data set from a software subsystem. The result shows that this approach produce statistically significant estimation.

**Xu, Ho & Capretz (2008)** in their research paper highlighted that object-oriented design has become a dominant method in software industry and many design metrics of object-oriented programs have been proposed for quality prediction, but there is no well-accepted statement on how significant those metrics are. In this study, empirical analysis is carried out to validate object-oriented design metrics for defects estimation. Approach: The Chidamber and Kemerer metrics suite is adopted to estimate the number of defects in the programs, which are extracted from a public NASA data set. The techniques involved are statistical analysis and neuro-fuzzy approach. The results indicate that SLOC, WMC, CBO and RFC are reliable metrics for defect estimation. Overall, SLOC imposes most significant impact on the number of defects. They concluded that the design metrics are closely related to the number of defects in OO classes, but cannot jump to a conclusion by using one analysis technique. They recommend using neuro-fuzzy approach together with statistical techniques to reveal the relationship between metrics and dependent variables, and the correlations among those metrics also have to be considered.

**Singh & Nisha (2009)** in their research paper the researchers examined 48 existing object oriented metrics proposed by various researchers to check their availability within design phase related to various constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. The study focuses on understanding and implementation of metrics in various phases of development. It shows that it is very intricate to determine which metric is more useful in which area. This study concluded that inheritance, information hiding, reusability and polymorphism metrics can be designated as design metrics but size, coupling and cohesion metrics are not considered as design complexity metrics as most of them are evaluated from source code or using information of later stages. This was very helpful as it worked as a pilot study.
Koru, Zhang, Emam & Liu (2009) in their study shown the relationship between the size and defect proneness of software modules. They analysed class-level size and defect data in order to increase understanding of this crucial relationship. They studied four large-scale object-oriented product: Mozilla, Cn3d, jBoss and Eclipse and observed that defect proneness increased as class size increased, but at a slower rate; smaller classes were proportionally more problematic than larger classes. So the developer should consider giving higher priority to smaller modules when planning focused quality assurance activities with limited resources.

Catal & Diri (2009) studied that software quality engineering comprises of several quality assurance activities such as testing, formal verification, inspection, fault tolerance, and software fault prediction. Until now, many researchers developed and validated several fault prediction models by using machine learning and statistical techniques. There have been used different kinds of software metrics and diverse feature reduction techniques in order to improve the models' performance. However, these studies did not investigate the effect of dataset size, metrics set, and feature selection techniques for software fault prediction. This study is focused on the high-performance fault predictors based on machine learning such as Random Forests and the algorithms based on a new computational intelligence approach called Artificial Immune Systems. They used public NASA datasets from the PROMISE repository to make their predictive models repeatable, refutable, and verifiable. The research questions were based on the effects of dataset size, metrics set, and feature selection techniques. In order to answer these questions, there were defined seven test groups. Additionally, nine classifiers were examined for each of the five public NASA datasets. According to this study, Random Forests provides the best prediction performance for large datasets and Naive Bayes is the best prediction algorithm for small datasets in terms of the Area Under Receiver Operating Characteristics Curve (AUC) evaluation parameter. The parallel implementation of Artificial Immune Recognition Systems (AIRS2Parallel) algorithm is the best Artificial Immune Systems paradigm-based algorithm when the method-level metrics are used.

Chahal & Singh (2009) in their study advocated that design of a software product largely influences its quality. Good design is one of the pre-requisites of a high quality product. Metrics are usually used to assess the quality of software designs.
The metrics for object oriented design focus on design characteristics, such as abstraction, coupling, cohesion, inheritance, polymorphism and encapsulation and are applied at attribute, method, class, pack-age, file and systems levels. Design metrics help the software de-signers to understand the problem areas in a design and to develop prediction models. A number of studies have modeled relation-ships between object oriented metrics and reusability, defects and faults, maintainability, and effort, and cost savings. So design metrics can give an early indication of goodness of design and thus of the software product developed using that design. If designers know symptoms of bad design then it is helpful for them to avoid the bad design. In this paper, we have explored some of the symptoms of bad design and studied metric relationships which high-light these symptoms. This was very much match with the work while organizing the study.

_Yadav & Khan (2009)_ in their research paper highlighted that reliability is an invention that all look for their daily life. They said that during the last decade a great deal of progress has been made in the development of fault tolerance techniques in order to improve the reliability of software. They proposed a mechanism using fault tolerance technique for achieving reliable software. The impact of unreliable software’s proneness on the software characteristics has been shown.

_Malhotra, Kaur & Singh (2010)_ stated that empirical validation of software metrics to predict quality using machine learning methods is important to ensure their practical relevance in the software organizations. It would also be interesting to know the relationship between object-oriented metrics and fault proneness at different severity levels. They build a Support vector machine (SVM) model to find the relationship between object-oriented metrics given by Chidamber and Kemerer and fault proneness, at different severity levels. The proposed models at different severity levels are empirically evaluated using public domain NASA data set. The performance of the SVM method was evaluated by receiver operating characteristic (ROC) analysis. Based on these results, it is reasonable to claim that such models could help for planning and performing testing by focusing resources on fault-prone parts of the design and code. The performance of the model predicted using high severity faults is low as compared to performance of the model predicted with respect to medium and low severity faults. Thus, the study shows that SVM method may also
be used in constructing software quality models. However, similar types of studies are required to be carried out in order to establish the acceptability of the model.

**Dallal & Briand (2010)** discussed class cohesion during the object-oriented design phase. In practice, assessing and controlling cohesion in large systems implies measuring it automatically. One issue with the few existing cohesion metrics targeted at the high-level design phase is that they are not based on realistic assumptions and do not fulfill expected mathematical properties. This paper proposes a High-Level Design (HLD) class cohesion metric, which is based on realistic assumptions, complies with expected mathematical properties, and can be used to automatically assess design quality at early stages using UML diagrams. The study was benefitted by this paper while noting the class cohesion at the design level.

**Singh, Kaur & Malhotra (2010)** conducted a study on KC1 data set to investigate the association between C K metrics and fault proneness of classes taking severity of faults into consideration. Study shows that machine learning methods are better than simple statistical methods for identifying the fault proneness of classes and further found that CBO, WMC, RFC and SLOC are reliable metrics in predicting the fault proneness and their capability varies with the severity of faults.

**Ghantasala (2011)** in this Ph.D thesis elaborately described the technical jargons related to the broad field ‘software metrics’ which are useful for the scholar. Those are as follows: optimization models for component selection in designing of modular software system, software engineering, software engineering layers, software development life cycle, software designing, software testing, software testing strategy, software reuse, commercial off-the shelf Software, component based development, software reliability, software reliability modeling, fault tolerance.

**Shaik et al. (2011)** discussed in their paper that object-oriented design and development have become popular in today’s software development environment. To produce high quality object oriented applications, a strong emphasis on design aspects, especially during the early phases of software, development is necessary. Design metrics play a vital role in helping developers to appreciate design aspects of software i.e., improve software quality and developer productivity. Metrics data provides a quick response about the software product quality to software designers
and managers. By analyzing the metric data, we can forecast the quality of the OO system. Through this paper, they are trying to edify about the OOD, metrics, quality and the relationship between these. In this paper, they provide empirical evidence underneath the role of OOD Metrics specifically a subset of the CK Metric suite.

*Catal (2011)* highlighted through this work that software engineering discipline contains several prediction approaches such as test effort prediction, correction cost prediction, fault prediction, reusability prediction, security prediction, effort prediction, and quality prediction. However, most of these prediction approaches are still in preliminary phase and more research should be conducted to reach robust models. Software fault prediction is the most popular research area in these prediction approaches and recently several research centres started new projects on this area. This study 90 software fault prediction papers published between year 1990 and year 2009 and then we categorized these papers according to the publication year. This paper surveyed the software engineering literature on software fault prediction and both machine learning based and statistical based approaches are included in this survey. Papers explained in this article reflect the outline of what was published so far, but naturally this is not a complete review of all the papers published so far. This paper helped the scholar to investigate the previous studies from metrics, methods, datasets, performance evaluation metrics, and experimental results perspectives in an easy and effective manner.

*Pandey, Ajeet Kumar & Shrivastava, Vivek (2011)* stated that early fault detection model using integrated and cost-effective test case prioritization. Regression testing is one of the important software maintenance activities that let the software tester to ensure the quality and reliability of modified program. Although the regression testing is expensive, it is necessary to validate the software after every modification. To reduce the cost of regression testing, software tester may utilize test cases prioritization techniques. One potential goal prioritization is to increase a test suite’s rate of fault detection. An improved rate of fault detection can provide earlier feedback on the system, enabling earlier debugging. APFD (Average Percentage of Fault Detected) metric is used to measure the test suite’s fault detection rate. This paper presents an integrated test case prioritization approach to increase the test suite’s fault detection rate. Three important factors such as program change level
(PCL), test suite change level (TCL) and test suite size (TS) are considered to prioritize test cases. Proposed approach is applied on different in-house programs to validate its accuracy. Model results are found to be promising when compared with optimal prioritization techniques which always results an upper bound of APFD values.

*Shanthi & Duraiswamy (2011)* through their paper stated that object-oriented design has emerged as a dominant method in software industry and many new metrics have been proposed for quality prediction of object-oriented programs, but the significance of those metrics is not yet confirmed. Software process control can be improved and high software reliability can be achieved if faults are predicted early in the software life cycle. Testing quality related issues of software has become critical with the increasing importance of the quality of software. Many authors have suggested theoretical validation followed by empirical evaluation using proven statistical and experimental techniques for evaluating in the field of usefulness and relevance of any new metrics. In this paper, they have presented an empirical validation of software quality metric suites on open source software for fault-proneness prediction in Object Oriented Systems. The three metrics used here are Chidamber and Kemerer (CK) Metrics, Robert C. Martin Metric Suite and McCabe’s Metric Suite. Using these metrics suite, the defects present in different versions of Rhino software have analyzed to predict the software quality by making use of the fault proneness concept. From the results and empirical analysis, it is clear that the different metric suites have different efficiency in faults prediction. With the aid of this empirical analysis, they suggested the software professionals to find out those metric suites that can predict faults while developing the quality metric software products using the OO approach. This helps the scholar in preposition of the design metrics to predict fault at early stage.

### 2.2 Conclusion

The chapter comprehensively included the design literature with root sources relevant to the study. At the beginning various facets under which the resources were categorized is mentioned. The scholar claims the resources considered for reviews is very much essential for the present work as well as for the future use.
References


Li, Xiaobin (2002). Software reliability measurement: a survey. Degree of Master of Computer Science at Concordia University, Montreal, Quebec, Canada.


