CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

With the ever increasing growth in the software development technologies and demand for high quality software the researchers have focused on various aspects of software quality. A critical analysis of the literature would provide information regarding what has been done in the same area which will lead to a significant investigation. Detailed and careful reading of the reviews of expert researchers would also promote greater understanding of the chosen topic, procedures, methods and algorithms and enable to frame useful hypothesis. This chapter of the thesis provides a detailed review of the literature, the related topics in the field of object-oriented design metrics and its related tools. The review of literature initiated to devise a new Software Assessment Model to assess the qualities of design at its early stages.

Software quality metrics and models

Bourque et al. (2002) presents a progress report on an attempt to identify and develop a consensus on a set of candidate fundamental principles. A fundamental principle is less specific and more enduring than methodologies and techniques. It should be phrased to withstand the test of time. It should not contradict a more general engineering principle and should have some correspondence with “best practice”. It should be precise enough
to be capable of support and contradiction and should not conceal a tradeoff. It should also relate to one or more computer science or engineering concepts. The proposed candidate set consists of fundamental principles which were identified through two workshops, two Delphi studies and a web-based survey. The implementation of quality should therefore begin with the specification of user quality requirements at the time of development only.

Valenti et al. (2002) identifies a set of quality factors that can be used to evaluate a CBA System using the standard ISO9126, which provides a general framework for evaluating a commercial off the shelf software without covering the specificity of the application domain. Thus, our effort has been mainly devoted to the elicitation of a set of domain specific quality factors for the evaluation of a Computer Based Assessment System. The quality factors of these sub characteristics cannot be measured directly, but must be defined in terms of objective features.

Georgiadou (2003) proposed Generic multilayered customizable software Quality Model (GEQUAMO). GEQUAMO encapsulates the requirements of different stakeholders in a dynamic and flexible manner so as to enable each stakeholder (developer, user or sponsor) to construct their own model reflecting the emphasis/weighting for each attribute/requirement. Using a combination of the Composite Features Diagramming (CFD) Technique developed by the author, and Kiviat diagrams a multilayered and dynamic model is constructed. Instances of models are presented together with the algorithm for the computation of the profiles.

Trendowicz & Punter (2003) investigate to which extent existing quality modeling approaches facilitate high quality software product lines. First, define several requirements for an appropriate quality model. Then, use those requirements to review the existing quality modeling approaches. From
the conclude from the review that no single quality model fulfills all of our requirements. However, several approaches contain valuable characteristics. Based upon those characteristics, propose the Prometheus approach. Prometheus is a goal-oriented method that integrates quantitative and qualitative approaches to quality control. The method starts quality modeling early in the software lifecycle and is reusable across product lines.

Ortega et al. (2003) employed designed a Boehm model prototype that reflects the essential attributes of quality. This model was evaluated using a method so it can be validated and also enhanced. The evaluation method consisted of: designing a survey, formulating, validating and applying the measurement instruments; defining an algorithm to obtain the quality estimate and analyzing the results. The model prototype enabled the strengths and weaknesses of the software products studied to be identified. When evaluating a software product using the model prototype, it was possible to ascertain its compliance with the standards and use the results to improve it. Since the evaluation was systemic, processes that affect certain characteristics of the product could be identified. Companies can benefit from the model proposed because it serves as a benchmark that allows their products to evolve and be competitive.

Huynh et al. (2003) proposed a Center Of Mass (COM) model to represent their view of software quality/dependability as a multi attribute and multi-stakeholder concept. They used utility to represent the values of quality attributes, weights to represent the importance of an attribute to a stakeholder through stakeholder survey. They drew the COM model for each stakeholder's quality needs and evaluate whether the current system quality satisfies all the stakeholders' expectations. However, this model doesn't provide a solution to avoid the one-size-fits-all metrics for stakeholders to define their software quality requirements.
Svahnberg & Wohlin (2003) propose a decision-support method to aid in the understanding of different architecture candidates for a software system. Propose a method that is adaptable with respect to both the set of potential architecture candidates and quality attributes relevant for the system's domain to help in this task. The method creates a support framework, using a multi-criteria decision method, supporting comparison of different software architecture candidates for a specific software quality attribute and vice versa, and then uses this support framework to reach a consensus on the benefits and liabilities of the different software architecture candidates and to increase the confidence in the resulting architecture decision.

Svahnberg & Wohlin (2005) present an empirical study of a method that enables quantification of the perceived support different software architectures give for different quality attributes. This in turn enables an informed decision of which architecture candidate best fit the mixture of quality attributes required by a system. The study concluded that achieving the highest level of quality against a model or set of measures does not ensure that a sufficient level of quality is enhanced when compared to other methods.

Rawashdeh & Matalkah (2006) concluded that the software quality models namely McCall, Boehm, ISO 9126 and FURPS have limitation of neither able to measure quality and nor able to associate with functionality of a product.

Seffah et al. (2006) reviewed existing usability standards and models and highlighted the limitations and complementarities of the various standards. An explanation about unifying these models into a single consolidated, hierarchical model of usability measurement was also presented. This consolidated model was called Quality in Use Integrated Measurement (QUIM). In the QUIM model there are 10 factors each of which corresponds
to a specific facet of usability. These 10 factors are decomposed into a total of 26 subfactors or measurable criteria that are further decomposed into 127 specific metrics. The rest of this work explains how a consolidated model, such as QUIM, can help in developing a more consistent approach to usability measurement.

Cote et al. (2007) concluded McCall and Boehm model focus on the product perspective of quality to the detriment of other perspectives and these models are primarily useful in a bottom up approach. The objective is to identify the requirements for a software quality model to be used as a foundation to Software Quality Engineering (SQE). In theory, ISO/IEC 9126 seems well suited for SQE. They proposed the characteristics of a quality model which is usable throughout the software life cycle and which embraces all the perspectives of quality. The model was proposed through the comparative evaluation of existing quality models.

Andreu’s & Tziakouris (2007) quality model is based on ISO 9126 which may be used for development and evaluation of original components and may be tailored according to the organization re-user and the domain needs of the targeted component. The quality model introduced can be tailored according to the organization-reuser and the domain needs of the targeted component. The proposed framework is demonstrated and validated through real case examples, while its applicability is assessed and discussed.

Sibisi & Van Waveren (2007) proposed a new framework for customizing software quality models in the text and it is further shown how this framework was applied in a real working environment in an attempt to quantitatively validate it. The results collected in the study showed that the framework could be used reliably in customizing a generic software quality model at characteristic level only. The deviations at sub-characteristic level
were due to unclear questions in the generated Generic Quality Questionnaire that resulted in misunderstandings. And the metrics used to create these questions were not fully tested for validity and reliability due to time constraints. Enhancements are discussed in the study and it is further shown how reliability can also be achieved at sub-characteristic level.

Zhang et al. (2008) presented the relation between complexity and autonomic characteristics are discussed. An autonomic evaluation model of complex software is proposed, which has three levels: complexity, autonomic characteristics and quality of software. In this model, fuzzy transformation matrix and weight matrix of each hierarchy in factor metrics are given, by which, evaluation result of each hierarchy and the final comprehensive evaluation result can be calculated. According to the result, autonomic maturity test of complex software system has the significance of application and practice.

Sharma et al. (2008) surveys a number of quality models for traditional and component-based systems and proposes a new model for CBS by proposing some new characteristics, which may be very relevant in the context of components. All the quality characteristics may not be of prime importance for an application to be developed for a specific domain. Therefore, it is necessary to identify only those characteristics / sub-characteristics which may have higher priorities over the others. The present work uses Analytical Hierarchy Process (AHP) to assign the weight values to the characteristics for the proposed model. These weight values are then used to evaluate the quality contribution of sub-characteristics, characteristics and then finally the overall quality of the component by using the appropriate metrics. This approach can be used to identify and select better quality component among several others which can be used in the final system.
Jamwal & Jamwal (2009) research on measuring Design quality using design patterns faced many issues related to existing software metrics and quality models. In this work design quality factors and some of these issues that should be taken into account in very large information systems will be considered and present approach to design quality assessment.

Barkmann et al. (2009) develop tools for metrics analysis of a large number of software projects (146 projects with ca. 70,000 classes and interfaces and over 11 million lines of code). Moreover, validation of design quality metrics should focus on relevant metrics, i.e., correlated metrics need not to be validated independently. Based on the statistical basis, identify correlation between several metrics from well-known object-oriented metrics suites. Besides, we present early results of typical metrics values and possible thresholds. It empirically concluded that validation of design quality metrics should focus on relevant metrics i.e., correlated metrics need not to be validated independently.

Behkamal et al. (2009) proposed a new a quality model for evaluation of B2B applications. First, the most well-known quality models are studied, and reasons for using ISO 9126 quality model as the basis are discussed. This model, then, is customized in accordance with special characteristics of B2B applications. The customization is done by extracting the quality factors from web applications and B2B e-commerce applications, weighting these factors from the viewpoints of both developers and end users, and adding them to the model. Finally, as a case study, ISACO portal is evaluated by the proposed model.

Kalaimagal & Srinivasan (2010) discusses a quality model called Q'Facto12 based on the ISO25000 quality standard that can be used to evaluate the quality of COTS components. The model is an upgrade of an
earlier model called the Q'Facto 10 model that was proposed by us. They provide an outline of the Q'Facto 12 model compares the model with the Q'Facto 10 model and highlights why the Q'Facto 12 model is better than the Q'Facto 10 model. An experimental study to demonstrate the use of the model has also been presented.

Upadhyay et al. (2011) presents a Software Component Quality Model (SCQM) by overcoming shortcomings of existing quality models. Based upon this end user can take decision upon selection, evaluation and ranking of potential component candidates and wherever possible attain improvements in the component design and development.

AL-Badareen et al. (2011) intend to discuss the characteristics of the software products that influence the users' satisfaction on design quality. Based on the well-known design quality models and the emotion of the software users, a model of design quality evaluation based on users’ views

Object-Oriented Method Metrics and Models

Laranjeira (1990) proposed a software size estimation model and claimed that the model has the potential for providing more accurate size estimates than existing methods which are not yet reliable enough to be consistently used with existing cost estimation models. Proposed method takes advantage of a characteristic of object-oriented systems, the natural correspondence between specification and implementation, in order to enable users to come up with better size estimates at early stages of the software development cycle. Through a statistical approach the method also provides a confidence interval for the derived size estimates. The relation between the presented software sizing model and project cost estimation is also considered
at software development cycle. Then notice that there is no obvious comparison between this model and the existing methods.

Li & Henry (1993) presented a research work which concentrates on several object-oriented software metrics and the validation of these metrics with maintenance effort in two commercial systems. Statistical analyses of a prediction model incorporating 10 metrics were performed. In addition, a more compact model with fewer metrics is presented.

Hong et al. (1993) presents a formal approach to the comparison of six object-oriented analysis and design methodologies. For each of the methodologies, a formal representation of it is constructed as a meta-process model and a meta-data model. These two Meta-models of a methodology represent the steps of the analysis and design, the concepts, and the techniques provided by this methodology. Based on this uniform representation, an extensive comparison of these six methodologies is then performed. The results are given as a set of tables in which the similarity and differences of these methodologies are exhibited. Adopting this formal approach should avoid errors caused by misunderstanding or misinterpretation of these methodologies. Consequently, an accurate and unbiased comparison is made possible.

Chidamber & Kemerer (1994) research addresses these needs through the development and implementation of a new suite of metrics for OO design. Metrics developed in previous research, while contributing to the field's understanding of software development processes, have generally been subject to serious criticisms, including the lack of a theoretical base. Six design metrics are developed, and then analytically evaluated against 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.

Bieman & Ott (1994) criticized Chidamber and Kemerer’s Lack of Cohesion in Methods (LCOM) metric as not being effective at distinguishing between partially cohesive classes. They demand cohesion measures that are sensitive to small changes to be able to evaluate the relationship between cohesion and reuse.

Archer & Stinson (1995) provide an overview of the merging of a paradigm and a process, the object-oriented paradigm and the software product measurement process. Taxonomy of object-oriented software measures is created, and existing object-oriented software measures are enumerated, evaluated, and placed in taxa. This report includes an extensive bibliography of the current object-oriented measures that apply to the design and implementation phases of a software project.

Basili et al. (1996) presents the results of a study empirically investigated the suite of object-oriented (OO) design metrics. More specifically, goal is to assess these metrics as predictors of fault-prone classes and, therefore, determine whether they can be used as early quality indicators. Based on empirical and quantitative analysis, the advantages and drawbacks of these OO metrics are discussed. Several number of OO metrics appear to be useful to predict class fault-proneness during the early phases of the life-cycle. Also they are better predictors than "traditional" code metrics, which can only be collected at a later phase of the software development processes.

e Abreu & Melo (1996) experimentally evaluated the impact of Object-Oriented (OO) design on design quality characteristics. A suite of Metrics for OO Design (MOOD) is adopted to measure the use of OO design
mechanisms. The study showed that OO design mechanisms such as inheritance, polymorphism, information hiding and coupling, can influence quality characteristics like reliability or maintainability.

Hitz & Montazeri (1996) are partly evaluated by applying principles of measurement theory. Using the Object coupling measure (Coupling between object classes (CBO)) as an example, it is shown that failing to establish a sound empirical relation system can lead to deficiencies of software metrics. Similarly, for the object-oriented cohesion measure (Lack of Cohesion in Methods (LCOM)) it is pointed out that the issue of empirically testing the representation condition must not be ignored, even if other validation principles are carefully obeyed. As a by-product, an alternative formulation for LCOM

Sherif & Sanderson (1998) reported the implementation of OO metrics on two software projects developed at the Jet Propulsion Laboratory (JPL). The two projects are The Micro Generic Controller (UGC) and The Sequence Generator (SEQGEN). Both projects use the object-oriented paradigm and the C++ programming language. OO metrics and other measures are implemented to analyze and compare the two projects.

The Weighted Methods per Class (WMC) metric is defined as the sum of the complexity of a class’ local methods and intended to count the combined complexity of local methods in a given class. Li (1998) claimed that the metric carrying two different units respectively and can be used with two intentions: 1) count of local methods, and 2) sum of the internal complexity of all local methods, but the problem is the number of local methods and the internal structural complexity of local methods are two independent attributes of a class and the dual interpretation of WMC metric might create a difficulties to the practitioner. Li (1998) proposed two new
metrics: 1) Number of Local Methods (NLM) and 2) Class Method Complexity (CMC) to measure the two attributes that the WMC intends to capture.

Nesi & Querci (1998) proposed a new complexity and size metrics for effort evaluation and prediction with respect to the most important metrics proposed for the same purpose in the literature. They were also reported the validation of those metrics.

Chidamber et al. (1998) concluded that, an analysis of a set of metrics is performed in order to assess its usefulness for 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 Effort Estimate on three commercial object-oriented systems are provided. The results suggested 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.

Tang et al. (1999) investigated the correlation between object-oriented design metrics and the likelihood of the occurrence of object-oriented faults. The empirical study is conducted on three industrial real-time systems that contain a number of natural faults reported for the past three years. The faults found in these three systems are classified into three types: object-oriented faults, object management faults and traditional faults. The object-oriented design metrics suite is validated using these faults. Moreover, a set of new metrics are formulated.
Briand et al. (2000) empirically explore the relationships between existing Object-Oriented (OO) coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing. It also provides an investigation and analysis strategy to make these kinds of studies more repeatable and comparable, a problem which is pervasive in the literature on quality measurement. It is shown that by using a subset of measures, accurate models can be built to predict which classes most of the faults are likely to lie in. When predicting fault-prone classes, the best model shows a percentage of correct classifications higher than 80% and finds more than 90% of faulty classes. Besides the size of classes, the frequency of method invocations and the depth of inheritance hierarchies seem to be the main driving factors of fault-proneness.

El Emam et al. (2001) investigated whether there is a confounding effect of class size in validation studies of object-oriented metrics and show that, there is reason to believe that such an effect exists. A detailed empirical methodology is described for identifying those effects. A study is performed on a large C++ telecommunications framework to examine if size is really a confounder. The dependent variable was the incidence of a fault attributable to a field failure (fault-proneness of a class). The metrics that are expected to be validated are indeed associated with fault-proneness.

Briand and Wust (2001) propose a practical, repeatable, and accurate analysis procedure to investigate relationships between structural properties and development effort. Results indicate that fairly accurate predictions of class effort can be made based on simple measures of the class interface size alone. Effort predictions at the system level are even more accurate as using Bootstrapping; the estimated 95 percent confidence interval for MREs is 3 to 23 percent. But, more sophisticated coupling and cohesion measures do not help to improve these predictions to a degree that would be
practically significant. However, the use of hybrid models combining Poisson regression and CART regression trees clearly improves the accuracy of the models as compared to using Poisson regression alone.

Bansiya & Davis (2002) discussed a hierarchical model for the assessment of high-level Design quality attributes in object-oriented designs. The model provides the evaluation of structural and behavioural design properties of classes and objects and their relationship using a suite of object-oriented design metrics. The design properties such as encapsulation, modularity, coupling and cohesion are related to high-level quality attributes such as reusability, flexibility and complexity were empirically validated. The relationship between the design properties and quality attributes were weighted in accordance with their influence and importance which can provide an insight to the practical quality assessment in the object-oriented designs.

Subramanyam & Krishnan (2003) provide empirical evidence supporting the role of OO design complexity metrics, specifically a subset of the Chidamber and Kemerer suite (CK metrics), in determining software defects. Our results, based on industry data from software developed in two popular programming languages used in OO development, indicate that, even after controlling for the size of the software, these metrics are significantly associated with defects. In addition, find that the effects of these metrics on defects vary across the samples from two programming languages-C++ and Java. The results believe that these results have significant implications for designing high-quality software products using the OO approach.

Condori-Fernandez et al. (2004) describes a measurement protocol to map the concepts used in the Object-Oriented (OO) method requirements model onto the concepts used by the COSMIC full function points (COSMIC-
FFP) functional size measurement method. This protocol describes a set of measurement operations for modeling and sizing OO software systems from requirements specifications obtained in the context of the OO-method. This development method starts from a requirements model that allows the specification of software functional requirements and generates a conceptual model through a requirements analysis process. The major contribution of this work is an extended set of rules that allows estimating the functional size of OO systems at an early stage of the development lifecycle. A case study is introduced to report the obtained results from a practical point of view.

Similarly, Costagliola et al. (2005) presented their class point, a Function Points (FP) like approach, named class point, which was conceived to estimate the size of object-oriented products. In particular, two measures are proposed, which are theoretically validated showing that they satisfy well-known properties necessary for size measures. An initial, empirical validation is also performed, meant to assess the usefulness and effectiveness of the proposed measures to predict the development effort of object-oriented systems. Moreover, a comparative analysis is carried out, taking into account several other size measures.

Gyimothy et al. (2005) describes the object-oriented metrics given by Chidamber and Kemerer to illustrate how fault-proneness detection of the source code of the open source Web and e-mail suite called Mozilla can be carried out. Checked the values obtained against the number of bugs found in its bug database - called Bugzilla - using regression and machine learning methods to validate the usefulness of these metrics for fault-proneness prediction. They also compared the metrics of several versions of Mozilla to see how the predicted fault-proneness of the software system changed during its development cycle.
Braz & Vergilio (2006) introduce two metrics, also based on Use Cases (UCs). The first one, named Use case Size Points (USP), considers the internal structures of the UC and better captures its functionality. The second one, named Fuzzy Use case Size Points (FUSP), considers concepts of the fuzzy set theory to create gradual classifications that better deal with uncertainty. Results from an empirical evaluation show the applicability and some advantages of the proposed metrics.

Zhou & Leung (2006) used 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. The results are 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.

Breesam (2007) primary purpose of this work is to analytically and empirically validate a 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 metrics.

Olague et al. (2007) empirically validated OO metrics suites for their ability to predict design quality in terms of fault proneness in Object-Oriented (OO) components. Empirically validation of three OO metrics suites for their ability to predict design quality in terms of fault-proneness: the
Chidamber and Kemerer (CK) metrics, Abreu's Metrics for Object-Oriented Design (MOOD), and Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). The ability of these three metrics suites to predict fault-prone classes using defect data for six versions of Rhino, an open-source implementation of JavaScript written in Java. It is concluded that the CK and QMOOD suites contain similar components and produce statistical models that are effective in detecting error-prone classes. It is also concluded that the class components in the MOOD metrics suite are not good class fault-proneness predictors.

Breesam (2007) validated analytically and empirically a set of metrics that could 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 metrics. Five inheritance metrics such as Depth of inheritance, Number of children, Abstract classes, Use of multiple inheritances, method inheritance are validated.

Xu et al. (2008) empirical analysis is carried out to validate object-oriented design metrics for defects estimation. 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. Results: 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. The design metrics are closely related to the number of defects in OO classes, but it cannot jump to a conclusion by using one analysis technique. 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.

Selvarani et al. (2009) empirically validated a set of metrics proposed by CK is performed to assess their ability in predicting the design quality in terms of fault proneness and degradation. They also proposed the design complexity of object-oriented software with Weighted Methods per Class metric (WMC-CK metric) expressed in terms of Shannon entropy, and error proneness.

Neelamegam & Punithavalli (2009) did survey on four object oriented quality metrics. In this study, the metrics for object oriented design focused on measurements that were applied to the class and design characteristics.

Thapaliyal & Verma (2010) evaluated two metrics Weighted Method per Class (WMC) and Coupling between Object Classes (CBO) of Chidamber and Kemerer metrics Suite. Empirical study and tried to find out the nature of relationship of these metrics with defects. In other words, it has been investigated whether these metrics are significantly associated with defects or not. For this taken samples of 50 Java classes of different projects developed by the final year B. Tech students under the guidance of faculty members having 4-10 years experience in object oriented programming. Deliberately taken different projects & tried to check if these metrics can really be reliable measurements for predicting defects when applied to inherently different projects.

Shaik et al. (2010) have done statistical analysis for object oriented software metrics on CK metric suite by validating the data collected from the projects of some students. Metrics data provided quick feedback for software
designers and managers. They found out that if appropriately used; it could lead to a significant reduction in cost of the overall implementation and improvement in quality of the final product.

Sastry & Saradhi (2010) tried to implement software metrics with aid of GUI & also analyzed relationships of metrics to determine quality and quantity of software attributes measured with regard of object oriented software development life cycle.

Marian (2010) constructed a software defect prediction models, the model constructing process allows deciding whether the calculated metrics are usable as defect predictors. The estimated number of defects for a Java class is the model output. The model output may be used to select classes where the estimated number of defects is on high level. Testers with a good defect predictor should be able to reduce their test effort by testing only 20% of files (Java classes) and they still should be able to find most of the defects (80%). Models constructed in this research are evaluated by counting the percentage of Java classes that must be tested in order to find 80% of the defects.

Kumar & Kaur (2011) highlights the object-oriented software metrics proposed in 90s’ by Chidamber, Kemerer and several studies were conducted to validate the metrics and discovered several deficiencies. Chidamber, Kemerer proposed six software metrics as Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object classes (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM). A new metrics suite for object-oriented programming proposed by Li (1998) includes Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract
Data Type (CTA), and Coupling Through Message Passing (CTM) as an alternatives to Chidamber and Kemerer metrics.

Kecia et al. (2012) carried out an experiment to evaluate the practical use of the proposed thresholds. The results of this evaluation indicate that the proposed thresholds can support the identification of classes which violate design principles, as well as the identification of well-designed classes. The method used in this study to derive software metrics thresholds can be applied to other software metrics in order to find their reference values.

Jain et al. (2013) concluded that the earlier studies have collected metrics, but all have not validated the metrics. An automation tool for measuring Predictive Object Points with more accuracy has been built. The tool and results of its application for Java Projects are presented and discussed. Moreover, the study further pin points that there is a need to identify the design quality metrics efficiently otherwise the developer will be confused in many metrics and many values thus purpose of evaluation of quality would be lost.

Abhijeet et al. (2013) developed a tool which would convert the given code into UML diagram to study an application's code and behavior and also shows dependency relation between various class. It analysis various suggested approach and implements a new approach for drawing class diagram and activity diagram.

Paloma et al. (2014) proposed the concept of relative thresholds for evaluating metrics data following heavy-tailed distributions. The proposed thresholds are relative because they assume that metric thresholds should be followed by most source code entities, but that it is also natural to have a number of entities in the “long-tail” that do not follow the defined limits.
They describe an empirical method for extracting relative thresholds from real systems. The proposed thresholds express a balance between real and idealized design practices. It can be used to control and monitor the technical debt in a system and to raise an alarm when the debt reaches a dangerous level.

Hamed (2015) proposed a reverse engineering model for object-oriented programs using concept lattice. This model is based on applying the concept lattice for object-oriented features, in the form of class and embedded representation, based on representing the relations between various programs elements in a lattice structure. This facilitate the reverse-engineering of a class for which the source file is not available. It also, discusses how the lattice and the embedded method representation can be used in order to efficiently read source files if available.

Plosch et al. (2016) proposed a novel approach based on measuring design best practices that closes the gap between the identification of design flaws and the support for improvements. An expert group six researchers captured a set of 67 design best practices that are implemented by the framework MUSE (Muse Understand Script Engine). For a first validation of MUSE, its measuring result is compared with QMOOD, which is an established metric-based approach for measuring the quality of object-oriented design. The qualitative assessment based on data from six versions of the Java tool jEdit shows that MUSE is better suited to guide improvements than QMOOD, e.g., for the design property encapsulation QMOOD indicates no substantial changes in the Design quality while the data provided by MUSE highlights that the encapsulation property of jEdit became worse over time and it helps to uncover design flaws directly from the source code.
Chavez et al. (2016) presented a model-based approach for testing whether or not an implementation satisfies the constraints imposed by its design model. This model-based testing approach aims to efficiently reduce the test input space while supporting branch coverage criteria. They developed a prototypical tool and applied it to the Eclipse UML2 projects to evaluate the approach’s ability to uncover inconsistencies.

Shruti et al. (2017), proposed a methodology to remove redundancy in datasets record values after which log transformation technique is used to calculate threshold value. The results of the transformation after removing redundancy are then used to conduct fault-proneness classification based on threshold values and compared against the results of fault classification based on threshold value after log transformation without removing redundancy in dataset record values. The fault classification after removing redundancy in dataset record values is better than fault classification without removing redundancy in dataset record values. The Table 2.1 shows the related works with comparison.

**Table 2.1 Related Works**

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Validation Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research scholars</td>
<td>Briand et al 1994</td>
</tr>
<tr>
<td>over years</td>
<td>Li &amp; Henry, 1993</td>
</tr>
<tr>
<td>about this</td>
<td>Victos R.Basili,</td>
</tr>
<tr>
<td>Metrics</td>
<td>Duraisamy and</td>
</tr>
<tr>
<td></td>
<td>Lionerl Ramaraj,</td>
</tr>
<tr>
<td></td>
<td>Briand and 2005</td>
</tr>
<tr>
<td></td>
<td>Walcelio .Melo,</td>
</tr>
<tr>
<td>Suite of Abstract Data Type (ADT)</td>
<td>Coupling and Cohesion</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td><strong>Type of Product</strong></td>
<td><strong>Ada</strong></td>
</tr>
<tr>
<td><strong>Dependant variable</strong></td>
<td><strong>Fault occurrence in ADT's</strong></td>
</tr>
<tr>
<td><strong>Statistical technique</strong></td>
<td><strong>Logistic regression</strong></td>
</tr>
</tbody>
</table>
2.2 SUMMARY

To sum up, first this chapter discussed the basis of object-oriented design needs like time pressure, changing requirements and the immaturity of object-oriented design. Also, it discussed about the characteristic of good object-oriented design and the role of metrics in CMM model. A deep literature review is also done about the history of software metrics and the classifications of metrics. The five different metrics proposed by pioneers’ like Chidamber and Kemerer, Lorenz and Kidd, Brito, Li and Henry were also discussed in detail.