Review of Literature

Structured Analysis & Design and Object Oriented Technique have much in common. Both the methodologies use similar modeling constructs and support the three orthogonal views of a system. The difference between SA/SD and Object Oriented approach is primarily a matter of style and emphasis. SA/SD organizes a system around procedure. In contrast, an Object Oriented design technique (such as OMT) organizes a system around real-world objects, or conceptual objects that may exist in the user view of the world. Most changes in requirement are changes in function rather than in the objects so changes can be disastrous to procedure-based design. By contrast, changes in functions are readily accommodated in an object oriented design by adding or changing operations, leaving the basic object structure unchanged.

Vessey and Conger (1994)
Vessey and Conger [1994] also compared the same three types of analysis methods: process-oriented (structured), data-oriented (Jackson System Development), and object-oriented. To conduct of this study the selection of the group included six students of software engineering having no practical experience in any analysis method, educated with same training in all three methods during a university course. They were then assigned to one of three groups (two students per group) and given analysis problems of similar level of complexity to solve using one of above-mentioned three analysis methods. A protocol analysis was performed by researchers and was found that it was more difficult to learn, understand and implement OOA for novice analysts than data-oriented analysis and data-oriented analysis much more difficult to understand and implement than process-oriented analysis. This study apparently contradicts the finding of Boehm-Davis and Ross that OO is easier to apply. While the methods used by developers in the Vessey and Conger study were almost identical to those in the Boehm-Davis and Ross study, the former study used students instead of experienced developers and used a much smaller sample size. Also, the students were not randomly assigned to groups.

Pennington, Lee, and Rehder (1995)
Later on a protocol analysis was performed by Pennington, Lee, and Rehder [1995] on a total of ten experienced, professional application developers. The group included three expert procedural application developers, four expert OO developers, and three novice OO developers with good
exposure of procedural developers). All above three groups with different combinations were given a relatively simple whirl meet scoring problem and were asked to create a complete design using their respective methods. Completed designs were assessed in terms of quality and the developers were assessed and evaluated on the basis of productivity. The outcome of this discovered that the designs prepared by the OO experts were more comprehensive but of course taken more time as compared to the procedural experts. Even though they took more time, during the assessment of overall design quality, the OO experts were graded more efficient than the procedural experts when. The study concludes that OO designs are of higher quality than procedural designs and take less time to complete.

**Hardgrave and Dalal (1995)**

In this sequence Hardgrave and Dalal [1995] conducted a study in labs of 56 advanced undergraduate MIS majors, all enrolled in a senior level course on DBMS, to compare two competing data modeling techniques: the extended entity-relationship (EER) model and the Object Modeling Technique (OMT) of Rumbaugh. The independent variables were modeling technique (OMT or EER) and complexity of the resulting model. The students were randomly assigned to one of four groups, which each group given a previously prepared, complete model to review (simple OMT, complex OMT, simple EER, and complex EER). The students in each group, who had already received training in the techniques, were provided with two additional one-hour lectures specifically on their respective models. All the members of the group were then asked to take a test on their understanding of the models and complete a follow-up questionnaire. The dependent variables were level of understanding (measured by the score on the test), time to understand (measured by the time to complete the test), and perceived ease-of-use (measured by item scores on a questionnaire). The results obtained from the experiments indicated that, for both the types simple and complex systems, OMT models were more quickly understood than EER models. However, no significant difference was found for the depth of understanding and the perceived ease of use of the two methods, regardless of task complexity. Thus, OO modeling techniques may be more quickly understood, but it may not be possible to completely understand, compared to data-oriented techniques. One possible limitations of this study is that it compares object-oriented to data-oriented modeling techniques. These two techniques are more closely related than object oriented and process-oriented techniques, and
thus differences in understanding or superficial ease of use may be difficult to perceive, and even if detected, less relevant to the concerns of many practitioners and researchers.

**Hrischuk (1995)**

Hrischuk describe an approach based on constructing an early prototype which is then executed to produce angio traces. These angio traces are then used to construct workthreads (also known as timethreads or use case maps [Buhr and Casselman, 1992],[Buhr and Casselman, 1994], [Buhr and Casselman, 1996]), which are analogous to execution graphs. Workthreads provide empirical information about traversal frequencies for data-dependent choices and loops.

**Boehm, Davis and Ross (1996)**

There have been a number of studies conducted on OOAD (1992-1996) which have made direct comparisons between OO and conventional methods. Boehm, Davis and Ross compared the quality of designs and solutions for various projects using three different types of systems development methodologies: procedural, data-oriented (Jackson Systems Development, or JSD), and object-oriented. While the results look somewhat encouraging for OO, the experimental design of the research has some serious limitations, primarily with the inability to control for the level of prior experience within the respective methodologies.

**Agarwal, Sinha, and Tanniru (1996)**

Agarwal, Sinha, and Tanniru [1996] performed a thorough experiment comparing the ability of novice analysts to correctly perform a requirements analysis using either a process-oriented (PO) or an object-oriented (OO) analysis methodology. A total of 43 undergraduate students (with no prior training or experience in any type of systems analysis) were randomly divided into two groups: a PO group consisting of 24 students (n=24) and an OO group consist of 19 students.(n=19). Each group was trained six hours in its respective analysis methodology - the DeMarco method for the PO group and the Coad and Yourdon [1991] method for the OO group. Individuals in each group were then presented with two problems to analyze - one problem was clearly more function strong (PO) while the other was more structure-strong (OO). According to the theory of cognitive fit, the PO group should perform better on the PO problem, while the OO group should perform better on the OO problem. The researchers found that the PO group had
significantly better overall performance than the OO group on the PO task, but that there was no
difference in overall performance between the two groups on the OO task. The researchers
concluded that PO methodologies should be easier for novices to learn than OO methodologies,
possibly because people may have a greater tendency to reason procedurally. Object-oriented
methods typically defer consideration of performance issues until detailed design or
implementation. Even then, the approach tends to be very general. There is no attempt to
integrate performance engineering into the development process.

Fenton and Pfleeger (1997)
According to Fenton and Pfleeger, the commonly held viewpoint that no metric is “valid” unless
it is a good predictor can undermine this rigor. An analogy would be to reject the usefulness of
measuring a person’s height on the grounds that it does not tell us anything about that person’s
intelligence. Over the last few years, a number of researchers and practitioners have discussed
and prescribed object-oriented software development approaches (e.g. Rational [2001]; Baudoin
and Hollowell [1996]; De Champeaux [1997]; Booch [1994]; Jacobson [1992]). More broadly,
Blum [1994] summarizes his work as software development process is a model for “the
progression from identification of a need in some application domain to the creation of a
software product that responds to that need”

Baldassari (1998)
Baldassari propose an integrated object-oriented CASE tool for software design that includes a
simulation capability for performance assessment. The CASE tool uses petri nets for the design
description language rather than the general methods described above, thus the design
specification and the performance model are equivalent and no translation is necessary.

Briand (1999); Bandi (1998); Henderson-Sellers (1996)
The first major “discovery” of this sort is that Smalltalk-style objects look a lot like abstract data
types. That is, perhaps we can identify object classes and object types. This identification
introduces a host of problems. What does strong-typing mean in this context? How can (static)
strong-typing cope with dynamic binding of variables to object instances? How can a type theory
for objects adequately cope with single/multiple class inheritance? Should class inheritance be
permitted to violate encapsulation by permitting subclass methods to access inherited instance variables? How can we understand type-casting and coercion in an object-oriented language? How can we exploit encapsulation to support heterogeneous objects, that is, environments in which different objects may be implemented in various languages with, say, a common object-oriented interface language? Various answers to these problems have been proposed, and even implemented. Service times are estimated. This differs from the approach described here in that scenarios are derived from prototype execution rather than from the design and the system execution model is then generated automatically from the angio traces. Much of the research, however, has been opportunistic and fragmented, without regard for concerns such as completeness of coverage, and has resulted in unnecessary repetition of efforts. Without the benefit of a birds-eye perspective, researchers in different groups have continued to “discover” and “propose” similar metrics.

**Wust, Daly, and Porter (2000)**

Wust, Daly, and Porter discovered that the frequency of method invocations and the depth of inheritance hierarchies are the major determinants of fault-proneness of resulting software classes defined at the class level were used as independent variables to predict the probability of fault-proneness in class code (the classes investigated were either developed from scratch or were extensive modifications of library classes). Univariate analysis revealed that increased levels of coupling and inheritance have a significant impact on fault-proneness of classes while cohesion does not. Multivariate analysis showed that models involving coupling and inheritance measures could be developed to automatically detect faulty classes with an accuracy rate approaching 90%.

**El Emam, Melo, and Machado (2001)**

This study focused only on those metrics that are available at the design stage. The measures involved two characteristics of OO design classes, coupling and inheritance (briefly explained above). The applications involved in the study were two consecutive releases of a commercial word processing program written in Java. Data were collected on faults reported by users of both versions so that classes could be identified either faulty or not. Design metrics were applied to all classes in both versions to find the relationships between measures of coupling and inheritance
and fault-proneness of the classes. A multivariate model including class size (number of attributes and methods), export coupling (number of times a class's methods are used by other classes), inheritance depth (the number of levels of inheritance for a class) measures resulted in an R-squared value of .24, with export coupling having the predominant influence on fault proneness.


According to Johnson, Richard A. [2002] Object-oriented systems development (OOSD) is viewed by many as the best available solution to the ongoing "software crisis." However, some caution that OOSD is so complex that it may never become a mainstream methodology. To settle the controversy requires high-quality empirical evidence. This paper surveys the most rigorous research on OOSD available over the past decade. A review of these empirical studies indicates that the weight of the evidence tends to slightly favor OOSD, although most studies fail to build on a theoretical foundation, many suffer from inadequate experimental designs, and some draw highly questionable conclusions from the evidence.

**Purao (2003)**

It is an educational non-profit organization to further the field. Although most of the common use of pattern until now has been at the design stage, many people believe that analysis patterns can contribute more to reusability and software quality than the other varieties.

**Kulak (2003), Larman (2004)**

The importance of analysis and design in comparison to the later stages of the development lifecycle has been widely described in the literature [Kulak 2003], [Larman 2004]. The significance of OOA/D in the OO process has been demonstrated by Kim et al in 1995. During their experimentation process, they observed that the initial representation of the problem was most important activity in shaping the level of abstraction and potential reusability of an implemented solution.
Hamza (2004)
Pattern is not created but discovered or identified from existing world. There are four main approaches that have been used to develop analysis patterns. They are the direct approach, specialization approach, analogy approach, and stability approach. The Fowler's (1996) analysis patterns are belong to the direct approach category. They are not generalized or abstracted further after they are identified. In the specialization approach, identified patterns are abstracted so that they are easier to apply themselves in similar and related applications than those patterns formed by direct approach.

Barclay (2004)
In practice, object-oriented design models have been less useful throughout the lifetime of software systems than they should be. Design models are often large and monolithic, and the structure of designs is generally quite different from that of requirements. As a result, developers tend to discard the design, especially as the system evolves, since it is too difficult to keep its relationship to requirements and code accurate, especially when both are changing. Another problem, related to learning OO, is that although object-orientation is arguably more representative of real world problems than functional decomposition, it does not always reflect the way in which humans think.

The proposed study in the proposed work is primarily confined to Object-Oriented Analysis and Design (OOA/D) activity of the software development lifecycle. The reason behind this is, if the process of analysis which is the initial and abstraction is incoherent, the following OOD produced, will not sufficiently resemble the original problem at either a logical or structural level. This will adversely affect all later steps in entire software-development lifecycle, leading them tedious and error-prone.

The 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. But there is a little understanding
of the empirical hypotheses and application of many of these measures. It is often difficult to determine which metric is more useful in which area. As a consequence, it is very difficult for project managers and practitioners to select measures for object-oriented systems. In this paper we 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.


Wang also conducted an experiment which included 32 undergraduate students without previous systems analysis training or experience. The subjects were arbitrarily divided into two groups. A training of five hours was given to one group on the data flow diagram (DFD) method, while the other group was trained for five hours on an object-oriented analysis method. The subjects were then offered with a mini-case in the process of MIS. It was interesting to observe that the OO group spent notable less time on their analysis of the problem and produced solutions that were considerably more accurate. After completing the analysis, the subjects responded to a questionnaire concerning their perceptions of the analysis method used. The OO group reported that the OOA approach was comparatively easier to learn, grasp and understand.

In another study, Wang again compared a structured approach of analysis (DFD) with object-oriented analysis (OOA) using two groups of undergraduate MIS majors having no experience. Students were randomly assigned to two groups, 24 in the DFD group and 20 in the OOA group. Each participant learned his respective analysis method and created analysis diagrams based on information in a mini-case study. The total time allowed for training and problem solving was 7.5 hours spanning several class sessions. The two dependent variables were the syntactic and semantic accuracy (in conveying system requirements) of the resulting analysis diagrams. Using ANOVA techniques, the results indicated that the syntactic accuracy for the DFD group was significantly greater in the early sessions, but that syntactic accuracy for the OOA group was significantly greater in the last session. However, there was no significant difference in semantic accuracy for the DFD and the OOA groups. It is evident from this study that the outcomes of the study contradicting the results of this researcher's previous study, this experiment concludes that OOA seems to be more difficult to learn and apply than DFD, and that OOA does not produce
solutions of higher quality. Another important benefit claimed after the study for OOAD is better and improved communication development team members, as well as between users and developers.

Booch (2007)
In fact Object Oriented Analysis & Design is considered to be the most critical activities in the process of software development. The benefits of this approach are well documented [Booch 1994, 2004, 2007], [Coleman 1994], [Cooke & Daniels 1994], [Jacobson 1992], [Rumbaugh 1991, 2004]. The design process offers various benefits including an early assessment of technical feasibility, correctness and completeness of requirements; management of complexity and enhanced comprehension; greater opportunities for reuse; and improved extensibility. There are number of approaches available for the software design, however in present context, the object-oriented design paradigm has become the standard approach throughout the software development process, but many issues remain open for research into improving its effectiveness against these benefits.

Apart from the fact that mental models provide a cognitive medium for interacting with complex situations, a key point of Norman’s statement is that mental models are formed. That is they are developed through prior experience and understanding, and are a dynamic construct that can be updated in the face of changing circumstances. Furthermore, mental models provide the basis for selecting or ignoring cues from the environment (or in fact even recognizing them at all) as well as determining how such information is interpreted and used within the model.

K P Srinivasan and Dr T Devi (2011)
This paper introduces a procedure that has been developed for evaluating an object-oriented design of a system that involves many classes. This approach involves two new metrics called Total Class Metric (TCM) and Total System Metric (TSM) that assess the design of a class and system as a whole respectively during object-oriented development process. In the increasing use of object-orientation in software development, there is a growing need to measure efficiency and effectiveness of the design process. In response to this need, a number of researchers have
developed various metrics for object-oriented systems. A procedure has been introduced for evaluating the effectiveness of the object-oriented design of a system for the improvement of the software process instead of using individual design metrics. The total class metric is defined based on a set of seven metrics which have been formulated using main attributes and significant characteristics of an object-oriented design of the system. This research paper discusses in detail about the new approach, total class metric and total system metric to represent the single quality value for the entire system design to judge the effectiveness of the design. These metrics will be useful in measuring object-oriented design and feedback system of software measurement thus yielding an effective object-oriented design.

During the past one and half decades (1996-2010), empirical studies of OOAD have shifted their focus from direct comparisons of OO and conventional methods to an exploration of the characteristics of OOAD that contribute to the quality of completed OO systems. This shift is likely due to the increased overall acceptance of OOAD, leading researchers away from comparisons to traditional methods.