Journal of Computer Science 7 (2): 143-147, 2011
ISSN 1549-3636
© 2011 Science Publications

Study of Object Oriented Analysis and Design Approach

[1]Sunil Kr Pandey, [2]G.P. Singh and [1]Dr. Vineet Kansal
[1]Department of I.T., Institute of Technology and Science, Mohan Nagar, Ghaziabad, U.P. India
[2]Department of Science Government Dungar College, Bikaner, Rajasthan, India

Abstract: Problem statement: Object and component technologies, rapidly maturing branches of information technology, have been becoming pervasive elements of systems development, especially the recently popular Internet applications and thus leading to increased complexity and at the same time broader range of applications. **Approach:** This needs to be understood in order to maximize its benefits and applications with consistent results. However, mainstream Object Oriented Systems Development (OOSD), consisting of Object Oriented Analysis and Design (OOAD) and Object-Oriented Programming (OOP), has a history of difficulties and is still struggling to gain prevalent acceptance. **Results:** There have been number of studies and experiments conducted by experts and researchers in the past which provides a solid base to take up this study and look into various intricacies present. There have been several studies and focused efforts in this direction which laid down the basis for a segment of people to form the opinion as "technology adoption is mostly the result of marketing forces, not scientific evidence" whereas there have been another segment that believes that object technology is "still long on hype and short on results ...". The gurus of OOSD continue to tout its vast superiority over conventional systems development, even to the extent of developing a unified software development process. **Conclusion:** The advocates of OOSD claim many advantages including easier modeling, increased code reuse, higher system quality and easier maintenance. It is well understood that analysis and design are extremely critical aspects of successful systems development especially in the case of OOSD. As the development of any successful information system must begin with a well-conceived and implemented analysis and design, this study will focus on the most recent empirical evidence on the pros and cons of OOAD.

Key words: Object Management Group (OMG), Extended Entity-Relationship (EER), Object Modeling Technique (OMT), Jackson Systems Development (JSD), methodology, fault-proneness, Unified Modeling Language (UML)

INTRODUCTION

The developments of object-oriented systems became possible with the proliferation of object-based and object oriented programming languages in the early 1980s. As is often the case, programming languages are developed long before the theory of how to use them effectively and efficiently. While small systems may be developed successfully without the aid of a formal system of analysis and design, larger industrial strength (Smeda *et al*., 2005) projects require a more systematic approach. OOD methods emerged in the mid-1980s and OOA methods in the late 1980s. An OOAD methodology consists of processes (methods describing "how to"), techniques (formalisms, models, notation) and, possibly, tools (e.g., CASE). Some of the more significant published methods of OOAD include those of Booch (1993) (Wirfs-Brock and Johnson, 1990; Briand *et al*., 1999), Rumbaugh *et al*., 1999; Shlaer, 1988, Pancake, 1995) (Coleman, 1994). (Briand *et al*., 2000; Aleksy *et al*., 2006; Jacobson *et al*., 1999; Mehta and Muttoo, 2006). In fact, the number of OOAD methods exploded from fewer than ten to more than 50 between 1989 and 1994. The field of OOAD has made particularly important strides in just the past few years with the development of the Unified Modeling Language (UML), the current standard graphical language for OO analysis and design. UML started as a unification of the Booch and OMT methods at Rational Corporation in 1994 and incorporated OOSE by 1996. The Object Management Group (OMG) accepted UML as a standard modeling language in November 1997 after widespread contribution from industry.

MATERIALS AND METHODS

OOA is the process of converting the real-world problem into a model using objects and classes as the

Corresponding Author: Sunil Kr Pandey, Institute of Technology and Science, Mohan Nagar, Ghaziabad

modeling constructs (Srivastava and Sabharwal, 2006). The objects identified from OOA are called semantic objects since they have meaning in the problem domain. An OOA model should ideally be understandable by application experts who are not programmers. OOD is the process of converting the problem model (from OOA) into a solution model based on objects. During OOD, new objects, not found in the OOA models, are added for implementation purposes. The implementation details of semantic objects are also added (short of writing actual code in the target OOPL). OOD can be executed at different levels such as class design, system design and program design. According to Booch OOD "encompasses the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design". Thus, OOD produces models of the proposed information system (the solution) rather than models of the real-world system (the problem).

Empirical studies in OOAD:

Early studies on OOAD: Many early studies of OOAD (1992-1996) made direct comparisons between OO and conventional methods. Boehm-Davis and Ross (1992) compared the quality of designs and solutions for various projects using three different types of systems development methodologies: procedural, data-oriented Jackson Systems Development (JSD) and object-oriented.

Vessey and Conger also compared the same three types of analysis methods: process-oriented (structured) data-oriented (Jackson System Development) and object-oriented. A total of six software engineering students, inexperienced in any analysis method, received the same training in all three methods during a university course. This study seemingly contradicts the finding of (Boehm-Davis and Ross, 1992; Herbsleb *et al*., 1995) and 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 (1992) study, the former study used students instead of experienced developers and used a much smaller sample size (n = 6 Vs. n = 18). Also, the students were not randomly assigned to groups.

Pennington *et al*. (1995) performed a protocol analysis on a total of ten experienced, professional developers. Three were expert procedural developers, four were expert OO developers and three were novice OO developers (who were, however, expert procedural developers). All three groups were given a relatively simple swim meet scoring problem and asked to create a complete design using their respective methods. Completed designs were judged in terms of quality while developers were evaluated on productivity. The results revealed that the designs of the OO experts were more complete but took more time compared to the procedural experts. Even though they took more time, the OO experts were graded more efficient than the procedural experts when overall design quality was considered. The study concludes that OO designs are of higher quality than procedural designs and take less time to complete.

Hardgrave and Dalal (1995) performed a lab study of 56 advanced undergraduate MIS majors, all enrolled in a senior level DBMS course, to compare two competing data modeling techniques: the Extended Entity-Relationship (EER) model and the Object Modeling Technique (OMT) of (Vijay and Manoharan, 2009) The independent variables were modeling technique (OMT or EER) and complexity of the resulting model. The results indicated that, for both 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 not more completely understood, compared to data-oriented techniques. One possible shortcoming of this study is that it compares object-oriented to data-oriented modeling techniques. These two methods are much more closely related than object oriented and process-oriented techniques, so differences in understanding or perceived ease of use may be difficult to detect and even if detected, less relevant to the concerns of many practitioners and researchers.

Wang performed an experiment using 32 undergraduate students with no previous systems analysis training or experience. The subjects were randomly divided into two groups. One group was trained for 5 h on the Data Flow Diagram (DFD) method, while the other group was trained for 5 h on an object-oriented analysis method. The subjects were then presented with a mini-case in management information systems analysis. The OO group reported that the OOA method was easier to learn and understand. The OOA method was also rated superior overall. This study confirms the results of several previously cited studies: OOA produces higher quality models more quickly than procedural analysis.

In a separate study, Wang again compared a structured method of analysis (DFD) with Object-Oriented Analysis (OOA) using two groups of inexperienced undergraduate MIS majors. Students were randomly assigned to two groups, 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 h 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. Apparently contradicting the results of this researcher's previous study, this experiment concludes that OOA appears more difficult to learn than DFD and that OOA does not produce solutions of higher quality.

Another important benefit claimed for OOAD is improved communication among development team members, as well as between users and developers (Garceau *et al*., 1993). In this research, several field studies were conducted using developers' timesheets, videotapes of meetings on design activities and semi-structured interviews with developers. Results indicated that when OOD methods are used, fewer spontaneous episodes of clarification occur. Also, planned summaries and walkthroughs occur much more often when using OOD. More attention was given to the reasons for specific design choices for the OO projects. OOD seems to encourage a deeper inquiry into the reasons underlying design decisions but less inquiry into the requirements. The increased number of planned summaries and walk-through could result if developers perceive a lack of understanding among peers. Thus, the study may indicate that OOD decreases one form of communication and increases another simply because it is new or more difficult to understand, not because it is easier or more natural.

Supporters of OOAD claim that thinking in terms of interacting objects rather than in terms of functions or procedures should be more natural to humans Davies, Gilmore and Green (Lin, 2007) set out to test the claim that OO decomposition of the problem domain is more natural to the ways of human cognition than functional decomposition. The results showed that expert subjects seemed to focus more on the functional properties of the code while the novice subjects tended to classify the code fragments according to important features of the OO paradigm (class membership, object similarity, or inheritance relations). According to them, the "results appear to suggest fairly clearly that functional information is of much greater importance to experts than is information about objects and their relations" (p. 242). The implication is that OO decomposition is not more natural for expert developers, as was expected by the researchers. Of course, an alternative explanation is that experts are simply more experienced with functional decomposition and tended to see the code fragments in that way.

Agarwal *et al.* (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 (n = 24) and an OO group (n = 19). Each group was trained 6 h in its respective analysis methodology -the (DeMarco, 1979; Coleman, 1994) method for the PO group and the (Coad and Yourdon, 1991; Gao *et al*., 2004) 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). 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.

More recent studies on OOAD: During the past few years (1996-2001), 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 OOSD, leading researchers away from comparisons to traditional methods.

Ishrat *et al*. (2010) discovered that the frequency of method invocations and the depth of inheritance hierarchies are the major determinants of fault-proneness (Ishrat *et al*., 2010) of resulting software classes. Existing measures of coupling (classes using methods or attributes in other classes), cohesion (methods within a classes using common attributes of the class) and inheritance (classes deriving methods from ancestor classes) defined at the class level were used as independent variables to predict the probability of fault-proneness in class code. 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%.

A similar study by (Xu *et al*., 2008) 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 as 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 (Xu *et al*., 2008).

A study by L. F. Capretz, 2004 (Pennington *et al*., 1995) took a different approach to investigating characteristics of OO designs, specifically design documents utilizing UML. The independent variable in this study is the type of reading technique used by individuals to detect defects in UML design documents for OO systems. The idea is for knowledgeable individuals to read design documents to detect defects prior to implementation of the designs. Results indicate that PBR is much more effective and efficient for UML documents of OO systems than CBR. This study contrasts a manual, human approach to defect detection at an early stage of design to an automated approach using metrics at a late stage of design.

RESULTS AND DISCUSSION

A total of 12 empirical studies representing some of the best available in the field of OOAD have been presented. In nearly every instance where studies were favorable to OOAD, higher quality and productivity were cited as primary benefits. On the other hand, nearly every negative result focused on the difficulty of learning OOAD or the inherent complexity of OO designs. These results are consistent with the anecdotal OO literature. In any event, the results suggest that while OOAD may be somewhat more difficult to learn than conventional methods, the effort spent in education and training may ultimately pay off in increased quality and productivity. Some studies discussed above present mixed results on other important OOAD issues. For example, the OO paradigm was found to be more natural for developers (Vijay and Manoharan, 2009) although the logical derivation of this conclusion from the data is highly suspect.

The conclusion that OOAD enhances communication (Muruganantham *et al*., 2010) may actually highlight a potential disadvantage of OOAD, i.e., the OOSD may be more confusing, thus causing an increased level of communication. Nearly all studies where only negative results were obtained stemmed from the use of inexperienced students as subjects. This suggests that learning can play a tremendous role in the effectiveness of OOAD. Students given only a few hours or weeks of training in OOAD should not be expected to perform OO tasks particularly well, especially given that OOAD may be somewhat difficult to learn. The conventional wisdom is that proficiency in OOAD may require six to 18 months of fulltime experience (Capretz, 2004). Thus, many of the negative results could be attributed to the types of subjects chosen and the amount of training provided.

CONCLUSION

Generally, studies often use inexperienced students as subjects. Such practices may be acceptable when the purpose of the research is to explore the difficulty of learning OOAD, but not when research questions focus on the quality and productivity of models or completed systems. Also, the question of learning OOAD may be even more critical to experienced procedural developers who may be forced by management to make the transition to OO, but no studies were found that specifically address this group. Another potential problem exists with studies that attempt to quickly train novice students in OOAD. Instructors at universities where such studies are conducted are likely to be significantly less experienced in the new OO methodologies than the more established procedural methodologies. This condition could result in less than optimum conditions for effectively and efficiently transferring complex OO knowledge, making it even more difficult for students to adequately learn OO. An ideal situation would be to collect detailed data on experienced individual developers or development teams who create identical complete real-world systems (perhaps of varying complexity) using both conventional and OO methods. Regardless of the particular research question involved, better experimental designs with tighter controls and larger samples could enhance validity. The obvious dilemma in this type of research is obtaining the cooperation of sufficiently large numbers of qualified subjects for laboratory or field studies. However, without adequate experimental designs, a quick resolution to the OO controversy will remain elusive.

REFERENCES

Agarwal, R., A.P. Sinha and M. Tanniru, 1996. Cognitive fit in requirements modeling: A study of object and process methodologies. J. Manage. Inform. Syst., 13: 137-162.

Aleksy, M., L. Köblitz and M. Schader, 2006. MEDiator: A tool for automatic management of event domains. J. Comput. Sci., 2: 535-541. DOI: 10.3844/jcssp.2006.535.541

Boehm-Davis, D. and L. Ross, 1992. Program design methodologies and the software development process. Int. J. Man Mach. Stud., 36: 1-19. DOI: 10.1016/0020-7373(92)90050-U

Booch, G., 1993. Object-Oriented Analysis and Design with Applications, 2nd Edn., Addison-Wesley Professional, Boston, ISBN-10: 0805353402, pp: 608.

Briand, L., E. Arisholm, S. Counsell, F. Houdek and P. Thevenod-Fosse, 1999. Empirical studies of object-oriented artifacts, methods and processes: State of the art and future directions. Empirical Software Eng., 4: 387-404. DOI: 10.1023/A:1009825923070

Briand, L., W. Daly and J. Wust, 2000. Exploring the relationship between design measures and software quality in object-oriented systems. J. Syst. Software, 51: 245-273. DOI: 10.1016/S0164-1212(99)00102-8

Coad, P. and E. Yourdon, 1991. Object-oriented Analysis. 2nd Edn., University of Minnesota, New York, ISBN: 0136299814, pp: 233.

Coleman, D., 1994. Object-oriented Development: The Fusion Method. 1st Edn., The University of Michigan, USA., ISBN: 0133388239, pp: 313.

DeMarco, T., 1979. Structured Analysis and System Specification. 1st Edn., The University of Michigan, USA., ISBN: 0138543801, pp: 352.

Gao, Q., L.J. Brown and L.F. Capretz, 2004. Extending UML-RT for control system modeling. Am. J. Applied Sci., 1: 338-347. DOI: 10.3844/ajassp.2004.338.347

Garceau, L., E. Jancura and J. Kneiss, 1993. Object-oriented analysis and design: A new approach to systems development. J. Syst. Manag., 44: 25-33.

Hardgrave, B. and N. Dalal, 1995. Comparing Object-oriented and extended-entity-relationship data models. J. Database Manag., 6: 15-22.

Herbsleb, J., H. Klein, G. Olson, H. Brunner and J. Olson *et al*., 1995. Object-oriented analysis and design in software project teams. Human Comput. Interaction, 10: 249-292. DOI: 10.1207/s15327051hci1002&3_4

Ishrat, R., R. Parveen and S.I. Ahson, 2010. Pattern trees for fault-proneness detection in object-oriented software. J. Comput. Sci., 6: 1078-1082. DOI: 10.3844/jcssp.2010.1078.1082

Jacobson, I., G. Booch and J. Rumbaugh, 1999. The Unified Software Development Process. 1st Edn., Dorling Kindersley, India, ISBN: 978-81-7758-315-1, pp: 512.

Lin, J., 2007. Mapping UML component specifications to jee implementations. J. Comput. Sci., 3: 780-785. DOI: 10.3844/jcssp.2007.780.785

Mehta, B. and S.K. Muttoo, 2006. JBOOM: Java based object oriented model of software configuration management. J. Comput. Sci., 2: 29-32. DOI: 10.3844/jcssp.2006.29.32

Muruganantham, S., P.K. Srivastha and Khanaa, 2010. Object based middleware for grid computing. J. Comput. Sci., 6: 336-340. DOI: 10.3844/jcssp.2010.336.340.

Pancake, C.M., 1995. The promise and the cost of object technology: a five-year forecast. Communi. ACM, 38: 33-49. DOI: 10.1145/226239.226247

Pennington, N., A.Y. Lee and B. Rehder, 1995. Cognitive activities and levels of abstraction in procedural and object-oriented design. Human Comput. Interaction, 10: 171-226. DOI: 10.1207/s15327051hci1002&3_2

Rumbaugh, J., G. Booch and I. Jacobson, 1999. The Unified Modeling Language Reference Manual. 1st Edn., Addison-Wesley Professional, Boston, ISBN-10: 020130998X, pp: 576.

Shlaer, S.,1988. Object-Oriented Systems Analysis: Modeling the World in Data. 1st Edn., Prentice Hall, USA., ISBN-10: 013629023X, pp: 144.

Smeda, A., T. Khammaci and M. Oussalah, 2005. Meta architecting: Toward a new generation of architecture description languages. J. Comput. Sci., 1: 454-460. DOI: 10.3844/jcssp.2005.454.460

Srivastava, N.P.S. and S. Sabharwal, 2006. The classification framework for model transformation. J. Comput. Sci., 2: 166-170. DOI: 10.3844/jcssp.2006.166.170

Vijay, J.F. and C. Manoharan, 2009. Initial hybrid method for analyzing software estimation, benchmarking and risk assessment using design of software. J. Comput. Sci., 5: 717-724. DOI: 10.3844/jcssp.2009.717.724

Xu, J., D. Ho and L.F. Capretz, 2008. An empirical validation of object-oriented design metrics for fault prediction. J. Comput. Sci., 4: 571-577. DOI: 10.3844/jcssp.2008.571.577

Wirfs-Brock, R.J. and R.E. Johnson, 1990. Surveying current research in object-oriented design. Communi. ACM, 33: 104-124. DOI: 10.1145/83880.84526