IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-5, Issue-9 E-ISSN: 2347-2693

Analysis of Applications of Object Orientation to Software Engineering, Data Warehousing and Teaching Methodologies

Biswajit Saha [1], Debaprasad Mukherjee [2*]

[1]Dept. CSE, Dr. B.C Roy Engineering College, Maulana Abul Kalam Azad University of Technology, Kolkata, India
[2]Dept. CSE and IT, Dr. B.C Roy Engineering College, Maulana Abul Kalam Azad University of Technology, Kolkata, India

**Corresponding Author: mdebaprasad@gmail.com*

Available online at: www.ijcseonline.org

Received: 07/Aug/2017, Revised: 18/Aug/2017, Accepted: 14/Sep/2017, Published: 30/Sep/2017

Abstract- Object oriented software engineering concepts are one of the most popular methods in the information technology industry and academia as well as in many other forms of engineering design. Software engineering is a field of engineering that came into existence owing to the various problems that developers of software faced while developing software projects. This paper analyzes some of the most important technological innovations in object oriented software engineering in recent times. The advancements in Object technologies that have been analyzed here include module coupling metrics, software up gradation, layered reusability, monitors, attribute objects, global software development contexts, software testing, recursive types, coupled and co-evolving classes, query-able source codes, meta-model for generating design alternatives, programming micro-worlds, design pattern detection, object schema migration, functional verification, work system theory, and state chart diagrams. From our analysis we predict further advancements in object technologies towards game development, metrics for software design analysis, addition to fundamental Object oriented programming language features and distributed software engineering.

Keywords: Object orientation, Software engineering, Data warehousing, Teaching methodologies.

I. INTRODUCTION

The object-oriented approach focuses on objects that represent abstract or concrete things of the real world. These objects are first defined by their character and their properties which are represented by their internal structure and their attributes (data). The behaviour of these objects is described by methods (functionality). Objects form a capsule which combines the character to the respective behaviour. Objects should enable programmers to map a real problem and its proposed software solution on a one-to-one basis, an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self").There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type. Many of the most widely used programming languages are multi-paradigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include Java, C++, C#, Python, PHP, Ruby, Perl, Delphi, Objective-C, Swift, Common Lisp, and Smalltalk.

Object-oriented programming techniques do not necessarily depend on object-oriented programming languages. However, the efficiency of object-oriented programming depends directly on how object-oriented language techniques are implemented in the system kernel. Object-oriented tools allow one to create object-oriented programs in object-oriented languages. They allow to model and store development objects and the

relationships between them. The object-orientation modeling of a software system is the most important, most time-consuming, and most difficult requirement for attaining the above goals. Object-oriented design involves more than just object-oriented programming, and provides logical advantages that are independent of the actual implementation.

There are three major features in object-oriented programming: encapsulation, inheritance and polymorphism. Encapsulation refers to the creation of self-contained modules that bind processing functions to the data i.e. tight coupling or association of data structures with the methods or functions that act on the data. These user-defined data types are called "classes," and one instance of a class is an "object." Encapsulation ensures good code modularity, which keeps routines separate and less prone to conflict with each other. Classes are created in hierarchies, and inheritance allows the structure and methods in one class to be passed down the hierarchy, and also the ability for a class to extend or override functionality of another class. That means less programming is required when adding functions to complex systems. If a step is added at the bottom of a hierarchy, then only the processing and data associated with

that unique step needs to be added. Everything else about that step is inherited. The ability to reuse existing objects is considered a major advantage of object technology. Object-oriented programming allows procedures about objects to be created whose exact type is not known until runtime. For example, a screen cursor may change its shape from an arrow to a line depending on the program mode. The routine to move the cursor on screen in response to mouse movement would be written for "cursor," and polymorphism allows that cursor to take on whatever shape is required at runtime. It also allows new shapes to be easily integrated. Data Protection is the ability to protect some components of the object from external entities. This is realized by language keywords to enable a variable to be declared as private or protected to the owning class. Interface means a definition of functions or methods, and their signatures that are available for use to manipulate a given instance of an object.

The advantages of object-oriented programming and software engineering are many. Through OOSE Complex software systems become easier to understand, since object-oriented structuring provides a closer representation of reality than other programming techniques. In a well-designed object-oriented system, it is possible to implement changes at class level, without having to make alterations at other points in the system. This reduces the overall amount of maintenance required. This is one of the most important advantages of OOSE/OOP. Through polymorphism and inheritance, object-oriented programming allows you to reuse individual components. This vastly reduces the development time of OO software. In an object-oriented system, the amount of work involved in revising and maintaining the system is reduced, since many problems can be detected and corrected in the design phase.

Areas of application of OO concept include real time systems design, simulation and modeling system, object oriented database; object oriented distributed database, client-server system, hypertext, hypermedia, neural networking and parallel programming, decision support and office automation systems, cim/cad/cam systems, artificial intelligence and expert systems, etc.

Some of the major directions that object oriented researchers are currently investigating are programming languages, concurrency and distribution, object management, software management and user environments.

Section I contain the introduction of analysis of applications of object orientation to software engineering, data warehousing and teaching methodologies, Section II contain the survey of recent literature in the related domain, Section III concludes with a brief synopsis of some of the recent advancements in object technologies and OOSE and unearth that certain important progress has been made in software design.

II. SURVEY OF RECENT LITERATURE

In this section a survey of recent literature is being carried out that will give an idea to the interested readers about the many diverse research fields of object orientation.

Bidve et al [1] used source code from 25 Java projects to analyze values esp. threshold values of 7 coupling metrics between different modules and their impact on object oriented software quality, to conclude that controlling/maintaining the values of parameter, inheritance and data abstraction during software development leads to better values of the quality attributes (e.g. reliability, integrity, maintainability) of software.

Sharma [2] proposed an approach based on inheritance and encapsulation for estimation of efforts for up gradation of object oriented programming system as per given requirement change, utilizing the degree of inheritance to identify methods that need either modification or reuse.

Baas in his doctoral thesis provided support for concept of reusability of object oriented components to improve productivity, testability and maintainability of integrated real-time software systems, by using a layered structure of reusable components based on the generality and specificity of the components to the application domain of the software, and the inter-component communication through OSI/CMIP conformant software highway [3].

The design of object oriented software systems needs the analysis of specifications for making decisions on the architecture of the software. The analysis of the specifications using simulation, for object oriented design of software architecture, involves the analysis of non-functional properties which can be measured using monitors. The authors showed how these monitors can be parametrically specified such that the instrumentation of the specifications can be automated and also the addition of monitors did not interfere with the behaviour of the software thus making it possible to have a library of monitors for the analysis of specifications [4].

Xia et al [5] developed a drawing application using attribute objects, which contain the attributes of digital content as UI objects, and can be manipulated (moved, cloned, linked, associated) through direct touch gestures such as simultaneous touch and pen input to quickly, coherently & consistently perform drawing interactions which are tedious or difficult.

Yoke eta al [6] designed and developed a proprietary role-playing interactive game for learning of object oriented programming and performed a case-study for game-based learning to conclude that this form of learning improves understanding and self-motivation towards object oriented programming.

Chandra and Singhal studied unit flow and data flow testing techniques and compared and contrasted them for software testing, quality assurance and avoidance of cost of software failure [7].

Jabangwe et al [8] developed a method to quantitatively evaluate quality during evolution of object oriented software, by analysing the impact of global software development context on the defects in the product. The authors used a method engineering approach to incrementally develop the method through its application to multiple industrial contexts. The authors claim that their proposal can help in inferences of development settings and make knowledge-based decisions during planning.

Holopainen studied different object oriented solutions for game development by testing in a game prototype built using Unity game engine by employing the principles of object inheritance with composition and the entity-component-system. The entity-component-system, concentrates on the creation of components from the object's attributes and functionalities. The solutions studied utilized the entity component system, inheritance with composition, and control of all similar entities by a greater manager entity, and the results were analyzed in terms of usability and versatility [9].

Maravic et al [10] compared knowledge in object oriented programming for college students through pencil-paper (control group) and computer adaptive test (experimental group) consisting multiple choice questions of constituting three clusters (easy, medium, and difficult). They found that students who worked on a computer-adaptive test achieved a higher average score than the students who did the traditional test.

Nierstrasz proposed "programming as a model" approach to long term evolution of object oriented software development where source code should encode a query able and modifiable model of the applications and utilizable by IDEs in these aspects. Furthermore, the author proposed that the boundaries between code, software and the software ecosystem should be transcended to facilitate the long term evolution of software systems [11].

Abdoli and Kara [12] used complexity capturing object oriented modelling in model based system engineering to develop coherent meta-model for generating design alternatives of the logical architecture of warehouses using the abstract interacting processes, sub-processes and resources.

Amato et al [13] proposed sharing analysis of objected oriented programs when two variables are bound to overlapping data structures, by using a graph theoretic representation of aliasing information (and thus encoding linearity and sharing too) and defining operators for analysis of an object oriented language like Java.

Micro worlds are used to give learners an intuitive and rapid understanding of fundamental abstract concepts of object orientation. Djelil et al [14] identified the design principles of micro worlds and explored each principle through the relation of the micro worlds to learning and object oriented programming. They also presented a novel programming world designed using the existing micro worlds.

Furia et al. [15] presented a modern auto-active semi-automatic verifier for functional verification of object oriented sequential programs with complex functional features. The verifier takes code with annotations as input and supports methodology for framing and class invariants. The authors also present the features of interface, design, implementation and performance on standard problems for the auto-active verifier.

Sunitha and Samuel [16] presented an implementation pattern for state diagrams (of UML) which include the two main components of state chart diagram that cannot be effectively implemented in object oriented way because of lack of proper one-is-to-one mapping i.e. hierarchical and concurrent states. This would facilitate automatic generation of skeletal code from state diagrams of systems designs of various event driven systems.

Börstler et al [17] proposed a readability measure SRES for object oriented programs as a proxy for understandibility, elegancy and simplicity from a cognitive and measurement perspective. They find that their readability measure is significantly correlated with quality measures, it is less sensitive to commenting and whitespace, and improves maintainability of programs.

Alter and Bolloju [18] proposed that ideas from work system theory and work system method can be a front-end to object oriented analysis and design, a step before creating use-case diagrams and UML artefacts, and thus providing path from business-oriented descriptions to formal, technical specifications to improve work systems performance.

Live programming requires real-time (run-time, development-time) support for object schema migration which is hindered by need of extra-levels of indirection or a very slow operation of sweep of all objects in the heap, during the exchange of object identities. Miranda et al [19] proposed Spur, a new object representation and memory management system which uses direct pointers but still sweeps the heap of object quickly by using forwarding objects and a partial read barrier avoiding the cost of explicit checking on the vast majority of object accesses.

Rashmi Chhabra et al [20] illustrated a system for designing of a relational schema from an object model which was represented in their UML form and finally transformed it into data warehouse.

Murat Oruc et al [21] presented a graph-mining approach for detecting object oriented design patterns for understanding the intent and design of software projects. The approach is based on probing input design patterns in the space of model graph of the source code by isomorphic sub-graph search process.

Parashar et al [22] presented a framework for mining the stream of class-change commits that are continuously generated during software development and evolution, using empirically validated changeability measures, to understand how classes have co-evolved or are change-coupled, and thus predicted the future changeability behaviour of classes for better maintenance of software.

Ghoreshi et al [23] proposed an incremental method for extracting test cases from formal object-oriented specifications which tries to overcome the deficiencies of existing methods of not being aligned with the iterative, incremental, adapting and evolving nature of object oriented development process.

Finding proper sample applications that help developers to mitigate problems while using object-oriented application frameworks like large and complex APIs and often incomplete user manual, is a hard and time- consuming challenge. Noei et al [24] tried to address this problem and propose EXAF which is a search engine for finding sample applications of object-oriented framework-provided concepts.

In object-oriented programs, objects often provide methods whose parameter types or return types are the object types themselves, but such explicit recursive types, which lead to a mismatch between sub classing and sub typing, are not supported in most object oriented programming languages. The authors thus proposed an approach to support methods with This Types through a new encoding of objects that allows sub typing by sub classing even in the presence of negative occurrences of type recursion variables, by distinguishing object types from existential object types, and ultimately they formalized language features to support This Typed methods. Ryu [25] further produced an open-source implementation of the approach as an extension of the Java programming language.

III. CONCLUSION

The object-oriented approach focuses on objects that represent abstract or concrete things of the real world. There is significant diversity of OOP languages, but the most popular ones are class-based. The efficiency of object-oriented programming depends directly on how object-oriented language techniques are implemented in the system. Through OOSE complex software systems become easier to understand, since object-oriented structuring provides a closer representation of reality than other programming techniques. We perform a thorough analysis of some of the recent advancements in object technologies and OOSE and find that certain important progress has been made in software design like coupling metrics between different modules, up gradation of object oriented programming system as per given requirement change, design of integrated real-time software systems, by using a layered structure of reusable components, attribute objects, meta-model for generating design alternatives, implementation pattern for state diagrams, real-time object schema migration, co-evolved or change-coupled classes, and detecting object oriented design patterns. In software analysis, we find that contributions has been made on unit flow and data flow testing, global software development context on the defects in the product, source code encoding of a query able and modifiable model of the applications, sharing analysis, functional verification of object oriented sequential programs, and analysis of non-functional properties which can be measured using monitors. Other improvements were made in game-based learning, game development, computer adaptive testing, design principles of micro worlds, readability measure, work system theory, live programming and extracting test cases from formal object-oriented specifications, and recursive types.

Power et al [26] compute the different metric values during the design phase of the entire life cycle of software development. Also a component diagram consisting of the components and their associations has been prepared using ArgoUML software tool.

There are as such no limitations of this survey but in future as more and more new areas of research related to the current domain emerge there will be lot of discussions related to such work and that is the scope of future improvement of this work.

REFERENCES

[1]. Bidve, V. S., and P. Sarasu. "Coupling Measures and its Impact on Object-Oriented Software Quality." Indian Journal of Science and Technology 9.21 (2016).

[2]. Sharma, Yashvardhan. "Effort Estimation for Program Modification in Object Oriented Development." International Conference on Computational Science and Its Applications. Springer International Publishing, 2016.

[3]. Baas, Andre. An object-oriented component approch to building real-time software systems. Diss. 2016.

[4]. Moreno-Delgado, Antonio, Francisco Durán, and José Meseguer. "Towards Generic Monitors for Object-Oriented Real-Time Maude Specifications." (2016).

[5]. Xia, Haijun, et al. "Object-Oriented Drawing." Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. ACM, 2016.

[6]. Wong, Y. S., Hayati, M. Y. M., & Tan, W. H. (2016, September). A Propriety Game-Based Learning Game as Learning Tool to Learn Object-Oriented Programming Paradigm. In *Joint International Conference on Serious Games* (pp. 42-54). Springer International Publishing.

[7]. Chandra, Alaknanda, and Abhishek Singhal. "Study of Unit and Data flow testing in object-oriented and aspect-oriented programming." 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH). IEEE, 2016.

[8]. Jabangwe, Ronald, et al. "A method for investigating the quality of evolving object-oriented software using defects in global software development projects." Journal of Software: Evolution and Process (2016).

[9]. Holopainen, Timo. "Object-oriented programming with Unity: Inheritance versus composition." (2016).

[10]. Cisar, Sanja Maravic, Petar Cisar, and Robert Pinter. "Evaluation of knowledge in Object Oriented Programming course with computer adaptive tests." Computers & Education 92 (2016): 142-160.

[11]. Nierstrasz, Oscar. "The Death of Object-Oriented Programming." International Conference on Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2016.
[12]. Abdoli, Shiva, and Sami Kara. "Designing Warehouse Logical Architecture by Applying Object Oriented Model Based System Engineering." Procedia CIRP 50 (2016): 713-718.
[13]. Amato, Gianluca, Maria Chiara Meo, and Francesca Scozzari. "Exploiting Linearity in Sharing Analysis of Object-oriented Programs." Electronic Notes in Theoretical Computer Science 322 (2016): 3-18.
[14]. Djelil, Fahima, et al. "Microworlds for Learning Object-Oriented Programming: Considerations from Research to Practice." Journal of Interactive Learning Research 27.3 (2016): 247-266.
[15]. Furia, Carlo A., et al. "AutoProof: auto-active functional verification of object-oriented programs." International Journal on Software Tools for Technology Transfer (2016): 1-20.
[16]. Sunitha, E. V., and Philip Samuel. "Object Oriented Method to Implement the Hierarchical and Concurrent States in UML State Chart Diagrams." Software Engineering Research, Management and Applications. Springer International Publishing, 2016. 133-149.
[17]. Börstler, Jürgen, Michael E. Caspersen, and Marie Nordström. "Beauty and the Beast: on the readability of object-oriented example programs." Software Quality Journal 24.2 (2016): 231-246.
[18]. Alter, Steven, and Narasimha Bolloju. "A Work System Front End for Object-Oriented Analysis and Design." International Journal of Information Technologies and Systems Approach (IJITSA) 9.1 (2016): 1-18.
[19]. Miranda, Eliot, and Clément Béra. "A partial read barrier for efficient support of live object-oriented programming." ACM SIGPLAN Notices 50.11 (2016): 93-104.
[20]. Chhabra, Rashmi, Parveen Kumar, and Payal Pahwa. "An approach to Design Object Oriented Data Warehouse." International Journal of Research and Engineering 3.3 (2016): 54-56.
[21]. Oruc, Murat, Fuat Akal, and Hayri Sever. "Detecting Design Patterns in Object-Oriented Design Models by Using a Graph Mining Approach." 2016 4th International Conference in Software Engineering Research and Innovation (CONISOFT). IEEE, 2016.
[22]. Parashar, Anshu, and Jitender Kumar Chhabra. "Mining software change data stream to predict changeability of classes of object-oriented software system." Evolving Systems 7.2 (2016): 117-128.
[23]. Ghoreshi, M., and H. Haghighi. "An incremental method for extracting tests from object-oriented specification." Information and Software Technology 78 (2016): 1-26.
[24]. Noei, Ehsan, and Abbas Heydarnoori. "EXAF: A search engine for sample applications of object-oriented framework-provided concepts." Information and Software Technology 75 (2016): 135-147.
[25]. Ryu, Sukyoung. "ThisType for Object-Oriented Languages: From Theory to Practice." ACM Transactions on Programming Languages and Systems (TOPLAS) 38.3 (2016): 8.
[26]. P.L. Power,M.P. Singh,Bharat Solanki, Jawwad Wasat Shareef, "Computation of External view based Software Metrics: Java Based Tool", International Journal of Computer Sciences and Engineering, Vol.5, Issue.8, pp.33-43,2017.

Authors Profile

Biswajit Saha completed Bachelor of Engineering in Computer Science and Engineering in the year 2002 and Master of Engineering in Software Engineering in the year 2004. He is currently teaching in the department of Computer Science and Engineering of an engineering college in Durgapur, India. He has more than 15 years of teaching experience at both the undergraduate and postgraduate level of engineering.

Debaprasad Mukherjee has graduated with a degree in Electrical Engineering in 2001 and was awarded PhD on systems theory (complex systems) in 2011. He is currently teaching in a college in India. He has published papers on diverse topics revolving around the theme of systems theory. His long term general and research interest is metaphilosophy.

