Object-Oriented Software Methodologies: Roadmap to the Future

Usman Ali Khan[1], I. A. Al-Bidewi[2] and Kunal Gupta[3]

**[1] Department of IS, FCIT, King Abdul Aziz University, Jeddah
Kingdom of Saudi Arabia**

**[2] Department of IS, FCIT, King Abdul Aziz University, Jeddah
Kingdom of Saudi Arabia**

**[3] Amity Institute of Information Technology, Amity University, Lucknow,
Uttar Pradesh, India**

Abstract

Software Development Methodologies have survived a never ending evolving era, ever since it first came in horizon. Amongst the several methodologies, only Object–Oriented Methodology has been able to see the dawn of the day. Object–Oriented Methodology survived all the critics as well as the rapid changes in the software development industry. But is it braced for the future yet? The paper is oriented towards the existing Object–Oriented Software Development Methodologies. A brief discussion involving their origin and focus of the methodologies is given followed by a review of UML. We discuss the various approaches taken up by various methodologies. The discussion facilitates for the key notes for the survival of Object Oriented Methodologies.

Keywords: *Object Oriented Methodology, Software Crisis, UML, Integrated Methodology, Agile Methodologies.*

1. Introduction

The Software Crisis was identified four decades ago. Various methodologies and models came forward to address the issue, but almost all perished and became extinct as they were unable to transform according to the much necessary change. Object Oriented Methodology emerged as a revolution some two decades ago. It displayed its versatile nature and adopted to encounter the rapidly changing Software Industry. Object Oriented Methodology evolved from Semi- Structured, Partly Object oriented to the Unified Model to the Integrated and Agile Methodology. Its true that Object Oriented Methodology have survived the harsh wrath of change, but on the other hand, it has been unable to provide the complete solution for the Software Crisis.

Software Crisis is real and it still exists. One reason may be because the way people have treated the methodologies, to use them for their own purpose. Some treat them as a mean, others as ends. A majority have been treating them as products, which makes them easier to sell. But on the downside, if methodologies are treated as products, they seem to clutter. They start having advertisement like descriptions, obscuring wrappings, which is inefficient while explaining its underlying process.

A much better way of viewing the Methodologies would have been to view them with the perspective of process, rather than with the perspective of modeling languages, as it can provide help to the user of the methodologies by providing them more information with respect of their context. The description of such a methodology might include the details of activities performed in sub process, and the order in which they are performed; as well as the concise description of the underlying modeling language used in sub process definition.

The discussion that follows will make a comparative study of the object-oriented software methodologies, along with other methodologies and then provides a detailed overview of process pattern, and process meta models, and the future of next generation methodologies.

2. Methodologies – Framework for the Development of the Modern Era Software:

The Software Development Methodologies have been viewed as the means for organizing the various methods of software development in a timely and orderly execution manner. Informally, Software Development Methodology has been termed as a collection of phases, procedures, rules, techniques, tools, documentation, management, and training that can be utilized for the development of a system. It basically comprises of a set of modeling conventions, comprising of a modeling language, and a process, which can provide guidance as to the order of the activities, and offers criteria for monitoring and measuring a project's activities. The modeling language aids in modeling the different aspects of the system, and the process determines what activities should be carried out in order to develop the system.

3. Object Oriented Methodologies: A brief insight

By 1990's, there were various methodologies in the Software Industry to design software products; but each and every one of them exhibited several limitations. In order to overcome these limitations, Object–oriented Methodologies were developed. Object–oriented methodologies for software development were specifically aimed at viewing, modeling and implementing the system as a collection of interacting objects, using the specialized modeling languages, activities and techniques needed to address the specific issues of the object-oriented paradigm. Several Object–oriented Software Development were developed in the evolution of the Object–oriented Paradigms. A brief overview of some of them is given below:

3.1 First and Second Generations of Object-Oriented Software Methodologies:

The first software development methodologies termed as object-oriented were in fact hybrid: partly structured and partly object-oriented. The analysis phase was typically done using structured analysis (SA) techniques, producing data flow diagrams, entity relationship diagrams, and state transition diagrams, whereas the design phase was mainly concerned with mapping analysis results to an object-oriented blueprint of the software. These methodologies were hence categorized as *transformative*. The second generation of object-oriented methodologies evolved from the first generation and appeared between 1992 and 1996. First and second-generation methodologies are collectively referred to as *seminal* methodologies, in that they pioneered the unexplored field of pure object-oriented analysis and design, and in doing so laid the groundwork for further evolution.

3.2 The Unified Modeling Language (UML):

Grady Booch, Ivar Jacobson, and James Rumbaugh. All three had developed their own methods, but collaborated to combine them into the Unified Method, the OMG announced plans for a standard OO notation, and in June of 1996 UML version 0.9 was released. UML version 1.1 was adopted by the OMG in November of 1997. With these initial UML releases, dozens of competing methodologies were replaced by the language- and method independent UML. Several factors contributed greatly to the widespread adoption of UML. First, UML is language independent. Second, it does not advocate nor require a particular method. Third, it is readily accessible as UML specifications are free for download and any company may join the OMG. The Object Management Group (OMG) is the body responsible for creating and maintaining the language specifications. They define UML as, "a graphical language for visualizing, specifying, constructing, and documenting the artifacts of object oriented software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software. The primary goals in the design of the UML were:

- To provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models. Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language. Encourage the growth of the Object Oriented tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.

UML was developed as a language that can be utilized for Modeling Object Oriented Systems and Applications, and provide them with more clarity by making them readable, and thus more understandable. This essentially means that UML provides the ability to capture the characteristics of a system by using notions. UML provides a wide array of simple, easy to understand notions for documenting systems based on the Object Oriented Design Principles. These notions are called the diagrams of UML. These diagrams provide the user with the means of visualizing and manipulating model elements. The underlying premise of UML is that no one diagram can capture the different elements of a system in it's entirely. The UML is just that. It "unifies" the design principles of each of these methodologies into a single standard language that can be easily applied across the board for all Object Oriented Systems. UML does not have any dependencies with respect to any technologies or languages.

3.3 Integrated Methodologies: Third Generation:

Methodologies in this category are results of integrating seminal methodologies and are characterized by their process-centered attitude towards software development, typically targeting a vast variety of applications. Integrations have resulted in huge methodologies, difficult to manage and enact. In trying to achieve manageability, some of them have gone to extreme measures to ensure customizability (RUP), others have turned into generic process frameworks that should be instantiated to yield a process (OPEN), and yet others have resorted to process patterns for customizability (Catalysis). It was frustration with these methodologies that ultimately caused the agile movement. Although unwieldy and complex, integrated methodologies have a lot to offer in terms of process components, patterns, and management and measurement issues. Furthermore, some of them propose useful ideas on seamless development, complexity management, and modeling approaches.

3.4 Agile Methodologies:

Agile methodologies first appeared in 1995. The once common perception that agile methodologies are nothing but controlled code-and-fix approaches, with little or no sign of a clear-cut process, is only true of a small—albeit influential—minority of these methodologies, which are essentially based on practices of program design, coding, and testing that are believed to enhance software development flexibility and productivity. Most agile methodologies incorporate explicit processes, although striving to keep them as lightweight as possible.

4. Object oriented Methodologies: The big leap on to the next generation

Object Oriented Methodologies have come a long way, and is still standing tall. In order to take big strides in the future, Object oriented Methodologies should incorporate the following within their structure:

- The advocates of Object oriented Methodology assumes that software should be developed according to a mental model of the actual or imagined objects it represent, i.e. it should focus on the real world. It should however provide for the uninterrupted exposition of logic in a more easy representation, more of a ordinary human language. This would lead to even poorly thought-out designs and decisions to be more subtle and obvious. The resultant model would bridge the gap between intuitive and formal models.
- Object-Oriented Methodology still provides a way or method of solving a problem. If it is to stay for longer duration, it needs to evolve itself as a technology which can address various issues in the modern era software development.
- In several cases, Object-Oriented methodologies have proved to be less fruitful as compared to Procedural Language. Amongst them are Economy of Execution, Economy of Small Scale Development, and Economy of Compilation. Object Oriented Methodology should scale themselves up for such shortcomings. A Significant difference in productivity between OOP and procedural development has to be achieved in the next era of Object Orientation.
- Object Oriented Technology has been poor in modeling time in a coherent manner. With the growing advent of real-time systems, it is imperative that Object-Orientation should evolve to model time and real time objects in a more decisive manner and thus aiding in the accurate design of real time systems.
- By its very nature, Object Orientation is anti-modular and anti-parallel. Which in the current scenario is a serious issue. With the growing popularity of parallel systems, it is of essence that Object Orientation should incorporate features which support parallel computing and applications.
- Currently, Object Orientation is unable to provide for interface specifications that are rich enough to cover all the phases of the design cycle. In component based development especially, non functional characteristics can be incorporated as a part of interface specification to overcome this limitation.
- Object Orientation provides for standardizing notations, which is often not sufficient to achieve effective methods and unambiguous communication amongst designers. With the advent of Opinion mining and profiling, these modeling notations maybe freely reinterpreted, which in turn weakens the value of notation as an effective communication vehicle and designing tool. This implies that a model written in one formalism could be ill formed in another formalism.
- It becomes difficult for Object Orientation to combine Heterogeneous system and depict their composite behavior. Object oriented methodology has to provide for embedding the

detailed models in question into a framework that can understand the model being composed.

- Object oriented should target towards being a model driven, context management, aspect oriented, service driven architecture. And should take the role of whichever is required at respective place.
- Although Object oriented Storage technology is being used in Cloud Computing, but the methodology is not sufficient for providing an impact structure for Web enabled Context aware systems and services. The main issue comes in the handling and distribution of context information efficiently. Furthermore, challenges like aggregation of context information in a structured format, discovery and selection of appropriate context services are key area, where Object oriented paradigm has to evolve and adopt in order to provide for a better framework and design for such system.

5. Conclusion:

The above review has resulted in a number of conclusions, which can be elaborated as follows. UML was developed in an attempt to standardize and integrate the methodologies into a single, comprehensible unit. Yet some of the limitations were still at large. The complexity and inconsistency was still there, which gave rise for the development of some agile and lightweight methodologies which actually followed a different path from modeling. But even they were not entirely successful and we witnessed the comeback of the old methodologies as well as new developments in methodologies emerged which did not adhere to UML conventions. The evolution suggested that in order to develop new methodologies and technologies, not only the capabilities of the old methodologies should be considered; but also the fact that they have to be developed with a more systematic approach in mind. Despite of the entire enhancement in the development of methodologies, a number of problem areas have been observed. The new integrated methodologies are more complex, to be efficiently being brought into the practice. They have lack of scalability, and lack of a specific, unambiguous process. Object Oriented Methodologies have evolved over a period of time, and despite of all the limitations they are still considered to be the pioneer when it comes to software development. Ongoing researches are aimed to further bring around an improved version, which can provide for compactness, extensibility, consistency, visible rationality, and traceability to requirement. Considering the motivations and the special circumstances surrounding methodologies mergers and development, engineering a methodology through integration can be one of the most appealing one. Disciplined Engineering and a systematic approach is desired for the extraction of prosperous potential of Object-Oriented Software Development.

References

[1] Aoyama, M. (1998a, April 19 - 25). Agile software process and its experience. Paper presented at the International Conference on Software Engineering, Kyoto Japan.

[2] Aoyama, M. (1998b). Web-based agile software development. IEEE Software, 15(6), 56-65.

[3] Bloomberg, J. (1999, October). Software Methodologies on Internet Time. Developer.com. Retrieved March 11, 2001, from the World Wide Web : http:// softwaredev.earthweb.com/java/sdjjavaee/article/0, ,12396_616711,00.html.

[4] Bloomberg, J. (2001, January). Using the RUP for Enterprise e-business Transformation. WaveBend Solutions. Retrieved April 7, 2001, from the World Wide Web: http://www.therationaledge.com/content/jan_01/f_rupent_jb.html.

[5] Coad, P. (1999). Feature-Driven Development. Object International. Retrieved April 8, 2001, from the World Wide Web: http://www.togethersoft.com/jmcu/chapter6.PDF.

[6] Cockburn, A. (2000, July/August). Selecting a project's methodology. IEEE Software, 17(4), 64-71.

[7] Cockburn, A. (2000, September). Balancing Lightness with Sufficiency. American Programmer. Retrieved March 11, 2001, from the World Wide Web:http://www.crystalmethodologies.org/articles/blws/balancinglightnesswithsufficiency.html.

[8] Cockburn, A. (2001). Just-In-Time Methodology Construction. Humans and Technology. Retrieved March 11, 2001, from the World Wide Web:http://www.crystalmethodologies.org/articles/jmc/justintimemethodologyconstruction.html.

[9] Cutter. (2000, October). Light Methodologies Best for E-business Projects. Cutter Consortium. Retrieved March 11, 2001, from the World Wide Web:http://cutter.com/consortium/research/2000/crb001003.html.

[10] DOJ (2000, March). The Department of Justice Systems Development Life Cycle Guidance Document. United States Department of Justice. Retrieved April 1, 2001, from the World Wide Web: http://www.usdoj.gov/jmd/irm/lifecycle/table.htm.

[11] LexiBot (2001). Our Technology - Results: The LexiBot Expression. BrightPlanet. Retrieved April 1, 2001, from the World Wide Web:http://www.brightplanet.com/technology/results2.asp.
[12] Lindvall, M., & Rus, I. (2000, July/August). Process diversity in software development. IEEE Software, 17(4), 14-18.
[13] Ubiquitous Cloud: Managing Service Resources for Adaptive Ubiquitous Computing
Koichi Egami, Shinsuke Matsumoto and Masahide Nakamura
[14] 1st IEEE PerCom Workshop on Pervasive Communities and Service Clouds
[15] Component-Based Design for the Future
Edward A. Lee and Alberto L. Sangiovanni-Vincentelli
IEEE

Dr. Usman Ali Khan is an Associate Professor at IS Department, Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, Saudi Arabia. He has done his Ph.D. IT (Object Oriented Software Design) in 2007. He is in the field of teaching and research since 1995. He has published fourteen research papers at national and international forums. He is teaching Software Engineering domain courses at graduate and undergraduate level from more than thirteen years.

Dr. Ibrahim A. Al-Bedewi is a Dean and Associate Professor at IS Department, Faculty of Computing and Information Technology, King Abdul Aziz University, Jeddah, Saudi Arabia. He has done his Ph.D. CS in 2000. He is in the field of teaching, research and administration since 2000. He has published many research papers at national and international forums.

Kunal Gupta is a Lecturer in Department of Information Technology, Amity University, India. He has done his M.Tech(IT) and MCA from India and is currently pursuing his PhD. He has published four International Papers. His teaching domain is Software Programming Languages. He has been in academics for more than seven years.