ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 5, November 2017

Comparative Study between Object-oriented Software Engineering and Agent-oriented Software Engineering

Maher T. Alasaady

***Abstract*— Object-oriented paradigm was used mainly within the last years, due to its features that could be exploited in software systems development. In the subsequent years, an agent paradigm was emerged to represent a new paradigm in software engineering with new properties for software systems development; therefore, the developers must make the decision to select the proper paradigm for development due the emergence of this new paradigm. From the important issues in this area is to identify the similarities and distinctions between these two paradigms, and determine the appropriate application size and domain for every single one of them. Since the existing problems such as lacking of differentiation between properties of these two paradigms, or at least not distinct between application size and domain that is commensurate with each them.**

In this research, we present and discusses the properties of object-oriented and agent focused paradigms, as well as the properties of object-oriented and agent-oriented software engineering, this accomplished in the form of a comparative study to determine the problems that the selected technology for appropriate application depended on it.

***Index Terms*—Software Agent, Object-oriented paradigm, Software Engineering, Software Agent & Multi-agents Systems.**

I. INTRODUCTION

Object Oriented (OO) technology has been used in an acceptable and increasing manner in the software development process, and has subsequently overcome for the other technologies used in software engineering and is still being used today. The reasons for its increase their use are the benefits of this technology, i.e. data abstraction, information hiding, encapsulation, concurrency [1]. And also because of the use of Object Oriented Software Engineering (OOSE), which in recent years has become a common approach commonly used by both Unified Modeling Language (UML) [2] and Unified Process (UP) [3].

However, despite the above benefits, we note some weaknesses of this technique, including the difficulty in learning, applying and practicing this approach [4] [5] [6]. On the other hand, may not be an effective tool for communication between customers and developers, especially in the initial stages of software development [7]. In addition, developers visualize OO concepts such as Object, Classes, and Inheritance, in a simple real-world format, but may mislead them from the real requirements of the software when they are from a formal perspective in the actual design process of these concepts.

To eliminate some weaknesses in OO technology, they have been combined with other technologies such as the use of Component Based, Patterns and Application Frameworks to take advantage of reuse and other aspects of software. However, there are many indications that OO cannot cope with the increasing complexity of software systems [8]. Examples of these complexities are the radical changes in the information systems environment, such as the use of the Internet, as well as the interactive, decentralized and independent nature of software applications at the present time, as well as the increasing number of interactions among subcomponents that are the result of increased system size. As a result of these complexities, it will be difficult to build large, complex and distributed systems using the OO approach with high quality and reliability, especially in the software product lines.

Agent Oriented (AO), is a new approach to software engineering, uses the concept of Agent as an essential way to analyze, design, and development software systems [9]. The benefit of Agent Oriented Software Engineering (AOSE) is its ability to transform agents' properties into important properties of complex systems, and to deal with components with high abstraction. One of the advantages of the Agency is the ability to reduce the interconnection of the components, which in turn leads to a fragmented, decentralized system of software, and its components can be modified [10]. On the other hand, AO technology is not used for small, central and some other systems; because it is intended for large, complex and decentralized systems. Software development using this technique is more complex and expensive. Therefore, it can be one of its weaknesses. It is necessary to define the target in advance to select the application domain and to determine the appropriate environment and the characteristics required for each one of these technologies.

II. RESEARCH OBJECTIVES & RELATED WORKS

The objective of this research is to study OO and AO techniques by comparing and defining their characteristics, as well as the characteristics of the developing process that used for each one of them, that is, software engineering. By discussing the results of the comparison and presenting the characteristics, it will be possible to describe the size and domain of the appropriate application for each one of these technologies, which will make it easier for developers to make the right decision for their choice.

Most of the researchers in this field such as [10], [11], [12], [13] compared the Object and the Agent and were interested in comparing their development methods and the appropriate programming language. They also discussed the autonomy and interactive property for each one of them. The methods of communication and modeling were also mentioned, but they did not mention all the properties that should have been mentioned, and they do not compare the software engineering architecture with full details for each one of them, and did not specify the application size and domain, in addition they do not use a scale and framework for the comparison.

III. OBJECT

The object is a threaded part of the data and operations it takes, and the Class can be considered as a template in which these objects can be configured and described [14]. Each application handles a certain number of entities or objects such as customer, product, invoice, etc. It will then be possible to create different categories for each of these entities. During the run time, the program will create one or more objects from each category. When this program wants to do something specific with one of these objects, it will call the function related to the object. Fig. (1) [3] illustrates the class model from which the object is generated.

IV. AGENT

The agent is a programmatic entity with intelligent characteristics such as independence, thinking, mobility, social ability, learning, cooperation and negotiation, which allows it to accomplish its work without the need for direct intervention or guidance by human or other entities, interacting interactively with other agents and with its environment to accomplish special tasks which cannot be accomplished by conventional software. Multi Agent System (MAS), where each agent in the system has incomplete information and capabilities for the purpose of solving the issues, therefore, each agent has a limited view, there is no general system control, the data is decentralized and processed be asynchronous [15]. Fig. (2) [6] shows the agent's interaction with its environment and depicts its behavior through internal state.

V. SOFTWARE ENGINEERING

Building high-quality software for applications is a difficult task in terms of the number of core components and its flexibility and interconnection. The role of software engineering is to provide models and techniques that make it easy to deal with this complexity. Software engineering is defined by [16]: "the application of systematic, disciplined and quantifiable approaches to the development, operation and maintenance of software, that is, the application of engineering to software".

Traditional software engineering techniques are generally based on the Waterfall Model, which consists of the following main phases: requirements, analysis, design, implementation, and maintenance [17]. The requirements stage is the definition stage of the high-level software specifications and core functions, what we want to achieve with the software, and the analysis of business requirements in looking at the functions of the software from the point of view of the end-user. Based on the results achieved in the requirements phase, the analysis of the software requirements is at a lower level and from the perspective of the software engineer. At the design stage, the minimum details of the software architecture, its main components and pseudo code are determined from algorithms and basic data structures, at this stage, optimal methods for software processing for the subsequent phase are identified in sufficient detail and are executable by programmers during the implementation phase. In the maintenance phase, errors, defects, bug fixes, new features, and software update are detected. The stages of program development that are sequentially written from one stage to another are illustrated in Fig. (3)

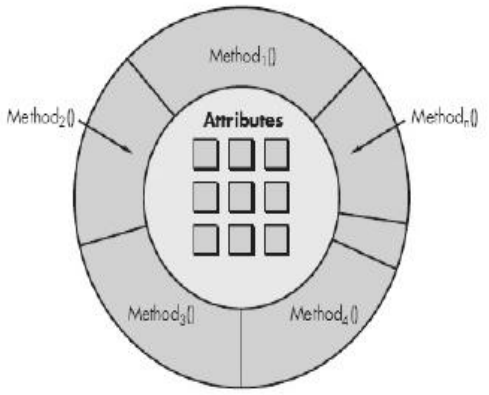


Fig. (1) Object model containing methods and attributes [3]

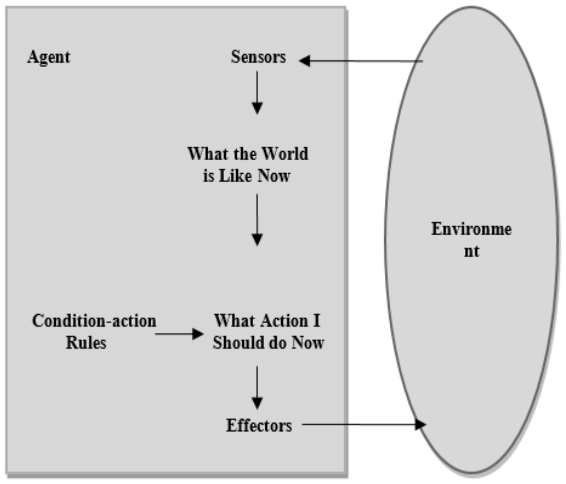


Fig. (2) A representation of agent-environment interaction [6]

VI. OBJECT ORIENTED SOFTWARE ENGINEERING (OOSE)

Object oriented technology has become used in software engineering as an alternative to a procedure oriented approach because of the properties that these organisms possess [18]. OOSE and other modern development processes are based on the classic waterfall with varying degrees of improvements, for example through iteration, and incremental development of software through multi-stage delivery [19].

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 5, November 2017

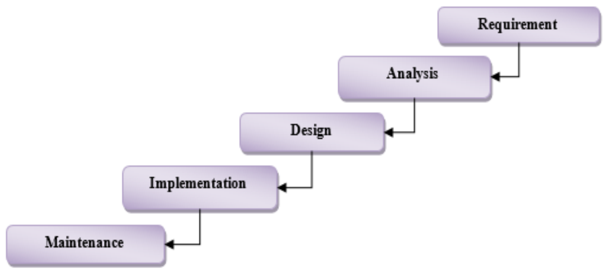


Fig. (3) Waterfall Model [19]

There are also alternatives to the waterfall model that combine the iterative and incremental and include various evolutionary models such as Spiral model, open source software development, agile methods and many other technologies.

The OOSE stages contain some of the techniques and tools used to accomplish their tasks. The Object-Oriented Analysis (OOA) contains the definition of the classes and objects that represent the case, the definition of how these objects interact with each other, the definition of attributes and operations for each object, and the communication mechanism that makes objects work with each other. One of the techniques used at this stage is Use Case (UC) scenarios that are dependent on the Actor who interacts with other entities. As well as Class-Responsibility-Collaborator (CRC), which translates UC information into classes and represents collaboration among other classes. The characteristics of these categories are then modeled using UML or another tool [1].

The Object-Oriented Design (OOD) design is divided into two phases: system design and object design. The system design defines a set of layers that perform specific functions of the system and defines which classes are encapsulated in the subsystems in each layer. The system design also defines three components: user interfaces, data management functions, and task management tools. The object design focuses on the interior details of each individual object, identifying the attributes, operations, and interactions in detail [3].

The Object-Oriented Programming (OOP) is used by converting the design into codes for an appropriate programming language such as C ++, java, C #, etc. The architecture interfaces and components are examined and tested in the test stage [19]. In addition, there are several software tools that help with each stage called CASE Tools.

VII. AGENT ORIENTED SOFTWARE ENGINEERING (AOSE)

It is one of the latest programming approaches and is a major transition in software engineering using a new approach that is more appropriate than object oriented technologies in the development of large, complex and distributed software systems. In order for the concept to be applicable, a uniform approach is required at all stages during the life cycle of agent development, which is similar to the unified process (UP) in the analysis and design of object oriented. This approach is called Methodology [20]. The main objectives of AOSE are to create methods and tools that have the potential to develop and maintain agent systems in an inexpensive, easy to use, flexible, scalable and high-quality manner, in other words, they will be quite similar in other branches of software engineering, i.e. OOSE.

AOSE follows the same steps as traditional software engineering techniques, but there are some techniques used at each stage of agent development that differ from OOSE techniques because of the different mechanisms of these entities [21]. These techniques differ in methods used to develop agents, each style has different techniques for development.

In the requirements stage, the requirements analysis and engineering methods such as *i [22] or KAOS [23] can be used. These methods are used to analyze system requirements and can be used as an alternative to the analysis stage as the two phases are integrated.

In the analysis phase, the majority of the methods used in the AOSE methods are to identify the roles played by components in the system that are detected at the requirements stage to be delegated to Agents, as well as to distinguish tasks, behavior, social, and communication protocols between agents. Schemes and models can be obtained as a product of this stage, as well as the overall system model which is represented with its relation to other components, and most likely at this stage the use of object-oriented methods to divide the system into subsystems or independent entities, as well as related UML schemes.

In the design stage, the methods of designing the components are used separately. The majority of methods are to distinguish the AgentType from which the agent is extracted, as well as the relationships between the agents, and to define the functions and methods in a structured and detailed manner.

In the development stage, programming tools such as JADE, JACK and Jadex are used, and the selection is based on design to suit it.

VIII. COMPLEXITY MANAGEMENT IN OOSE

Booch described that object-oriented software engineering that are used for complex systems should generally have the following mandatory characteristics [14]:

A. Decomposition in Objects

The ability to divide the large problem into smaller pieces to be more manageable and can be dealt with individually.

B. Abstraction in Objects

The ability to define a simplified model of the system that focuses on certain details and characteristics and leaves the other things, and therefore can focus attention on the salient aspects of this problem at the expense of less important details.

C. Organization in Objects

The process of defining and managing the interrelations between the different elements for the solution of the problem. The ability to define and activate organizational relationships helps designers to address complexity in two ways: first, by enabling a number of key elements to be grouped together and treated as a single high-level unit of analysis; and secondly, by providing a means to describe the high-level relationships between different units, i.e. a number of items may work together and collaborate to provide certain functions.

IX. COMPLEXITY MANAGEMENT IN AOSE

As Booch suggested, the principles of dealing with the complexities of object-oriented approaches are decomposition, abstraction, and organization. Jennings also suggests the same mechanism to demonstrate that the agent approach can also use these principles to build agents systems [24]:

A. Decomposition in Agent

The problem is divided into independent agents, and they can share high-level and flexible interactions. The independence of agents means that system engineering may be a group of specialized actors in resolving the problem and have control, and decide for themselves what actions should be at any time [8]. This natural representation of the complex systems being distributed and this decentralization reduces the complexity of system control, resulting in a lower degree of interconnection between the components.

B. Abstraction in Agents

In software design, stronger abstraction reduces the distance between the semantics of analysis units that are used intuitively to determine the concept of the problem and the construction found in the solution model. There is a clear and strong degree of similarity between the concepts of sub-systems and agent organization, since interaction between subsystems and their components can be seen as a high-level social interaction. For example, Booch [14] begins his analysis of complex systems from the following point of view: "At any given level of abstraction, a group of meaningful entities must cooperate for a higher purpose." On the other hand, we note that agents cooperate to achieve common objectives and coordinate their actions or negotiate to resolve contradictions that are at a higher level.

C. Organization in Agents

Agent approaches provide explicit representation of organizational relationships and structures. The entire subsystem can be viewed as a single component, team or group of agents. Different types of organizations, interactions and relations between agents can be defined. There are cooperative, competitive and utilitarian organizations. The agent's social property means that they can share complex and flexible interactions [9]. Agents are able to make decisions about the nature and extent of interaction at run time. This makes complex systems architecture easier for two reasons. First, in distributed and complex systems, potential interactions cannot be determined at a given time and for some reason, so the components need to be able to initiate and respond to interactions in a flexible manner, agents are originally designed to be able to interact, communicate and coordinate with one another, deal with unexpected situations and take timely decisions on the actions to be taken. The second reason is that all agents are active and continuous and that any coordination required is to deal from the bottom up through agent interaction, and this leads to a significant reduction in the management of the control of the relationships among software components.

X. OBJECT VS. AGENT

Although there are some similarities between object and agent, i.e. information hiding, interaction, but there are many differences between them, which can be illustrated by the following formula:

1) Objects are negative and the reason is that methods are called only when messages are sent by external entities [25].
2) Although objects encapsulate the state with behavior, but do not encapsulate the choice of movements that will be performed in the future [26].
3) OO technology may fail to equip a set of appropriate concepts and methods in the modeling of complex systems and at a high level of abstraction [8].
4) OO technology provides support but is simple to identify and manage organizational relationships between components [14].
5) Objects can use message passing to communicate with other entities [27].
6) Objects cannot interact independently with the environment [28].

g. OO technology can perform predefined movements through predictable events [14].

On the other hand, the agent technology provides the following characteristics:

1) Agents are characterized by self-control as well as the way they are called, so they do not need a central or top-down control because of their ability to manage and control themselves, which in turn makes agents act as active, proactive and not passive entities.
2) Agents are characterized by having their own goals, rules and plans that determine their decisions and where and when they do the acts. While objects do not have independent and flexible behavior, and agents can be considered independent entities because they have internal responsibilities [8].
3) Agent technology can manage complex systems through three main features to control these systems: Abstraction, Decomposition, and Organization [29].
4) Agents may be seen as interacting entities because of their ability to deal with messages that define the behavior and rules of their interaction.
5) Agents use language to communicate with each other, such as Agent Communication Language (ACL) [27].
6) Agents can interact with the open and unpredictable environment in a vital way through the proactive and Reactions [30].
7) Agents may fail to identify appropriate behavior for possible errors through unexpected events [26].

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 7, Issue 5, November 2017

Based on the theoretical comparison above, the formal comparison between OO and AO technology was carried out using feature-based technology [31] which uses the key concepts of infrastructure development in both OO and AO systems. Table (I) shows the process of comparing OO and AO technology with data chart represent the differences between them. The evaluation and comparison process were carried out by presenting these concepts in the scale of criteria for applying this concept, with a score of (1-7) assigned as a value that specifies that this criterion matches the corresponding technology. These grades are based on seven measures, in which case it will be possible to observe the possibilities and degrees of these methods that determine the viability, weakness or force of applying this method to the standard, As the higher scores indicate greater support for the standard.

XI. OOSE VS. AOSE

The comparison between object oriented software engineering and agent oriented software engineering was performed using the same method as the previous paragraph, a feature-based technique. Table (II) shows the OOSE and AOSE comparative process with data chart represent the differences between them.

Applications of agents' systems are appropriate in some cases where the requirements are complex and involve many distributed elements with different expertise and conflicting interests [32]. Otherwise, the choice is made in object technology. Agents' approach is best to be used in matters that are either physically or geographically distributed and where independent processes can be clearly defined. Examples of such issues are air traffic control systems, decision support systems, electronic commerce systems and other network and

Table (I) Comparison of OO and AO

	AO	OO
■ Abstraction	6	5
■ Mental Properties	7	2
■ Autonomy	7	1
■ Collaboration	6	3
■ Communication	6	4
■ Concurrency	5	5

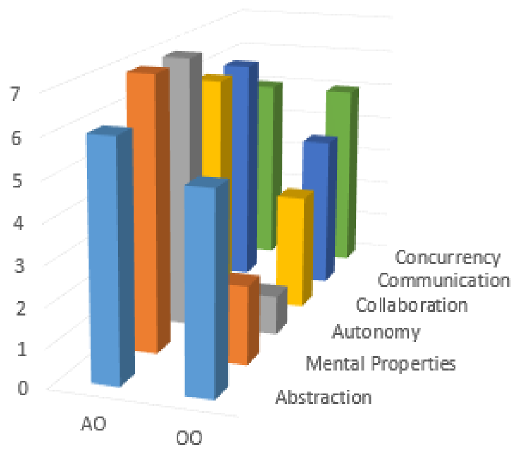
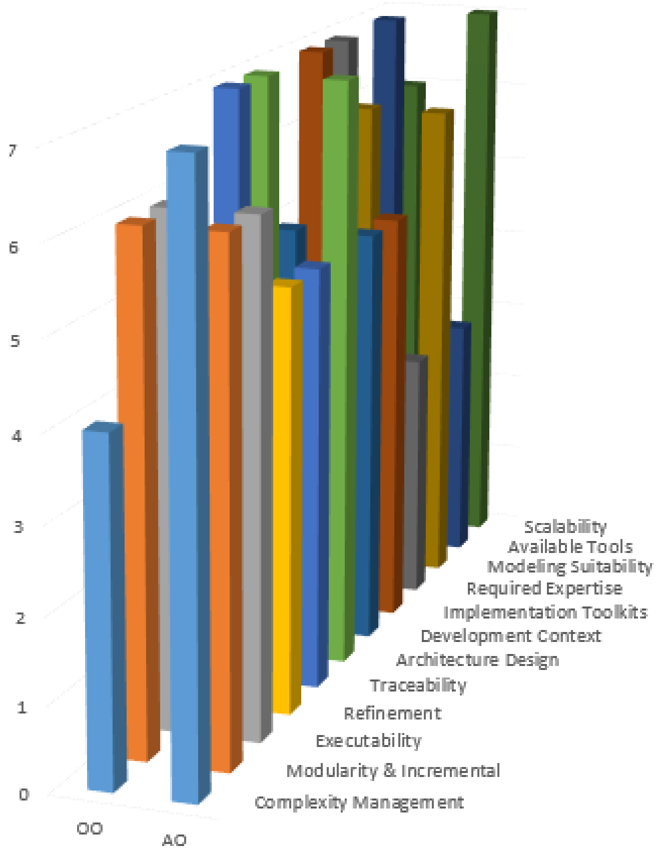


Table (II) Comparison of OOSE and AOSE

	OO	AO
■ Complexity Management	4	7
■ Modularity & Incremental	6	6
■ Executability	6	6
■ Refinement	5	5
■ Traceability	7	5
■ Architecture Design	7	7
■ Development Context	5	5
■ Implementation Toolkits	7	5
■ Required Expertise	7	3
■ Modeling Suitability	6	6
■ Available Tools	7	3
■ Scalability	6	7



XII. APPROPRIATE APPLICATION FOR BOTH OO & AO

Distributed control systems. But distribution is not a sufficient factor to use agent systems approaches, since there should also be basic needs for intelligence or adaptation in sub-processes that contain rational behavior.

But of the wrong solutions, is to make use of agent systems only because of complex requirements; there are some complex designs that do not require to use agent. Agent systems used rather than objects when targets and information are changing or conflicting and need to interact with each other effectively to achieve its objectives and perform its tasks. Also, it must be emphasized that in some cases where the central control is fundamental to only one agent or that we do not need any parallel actions, in this case we do not need to use agent systems approach.

There are some situations in which agent systems can be used, including the following: [33]

1) Applications requiring communication and interoperability between independent elements, expert systems, and decision-making systems.
2) Applications containing distributed issues with independent sources and information.
3) Issues that require solutions based on the diversity of different and distributed experiences, i.e. health care, for which the central agent cannot carry out his task without the assistance of other experts.
4) Issues in remote locations that need to understand interactions between communities and organizations.

Based on the above, we will compare OO with AO technique in terms of using the appropriate application type and size for each one of them. Table (III) shows the appropriate application size for both OO and AO technology. The percent is used to indicate the preference for using the technology for the corresponding application size.

In addition to determining the appropriate application size for each of the above techniques, a comparison was made between OO and AO technology to determine the appropriate application domain. Table (IV) shows the appropriate application domain for both OO and AO.

XIII. CONCLUSION & FUTURE WORK

By presenting the properties of both object and agent, the properties of the software engineering methods that are applied to each of these entities are discussed, and the method of dealing with the complexity of the systems is discussed in each of these techniques. In addition to comparing object and agent entities, it was concluded that:

Table (III) Appropriate application size for OO & AO

	OO	AO
🟦 Small Systems	95%	20%
🟧 Normal Systems	90%	50%
⬜ Large Scale Systems	80%	70%
🟨 Complex Systems	70%	90%
🟦 Distributed Systems	60%	95%

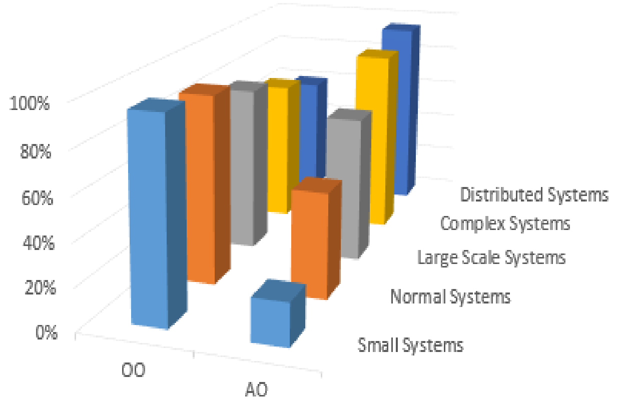
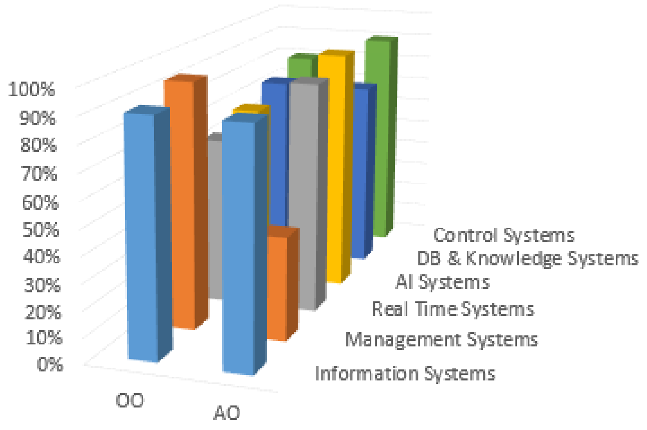


Table (IV) Appropriate Application Domain of OO & AO

	OO	AO
🟦 Information Systems	90%	90%
🟧 Management Systems	95%	40%
⬜ Real Time Systems	65%	90%
🟨 AI Systems	70%	95%
🟦 DB & Knowledge Systems	75%	75%
🟩 Control Systems	80%	90%



1) Agent technology can be used in complex and large-scale systems, because of their ability to deal with complexity and size increase, while the process of agent systems development is not easy, so it is a wrong solution to build small and medium-sized systems using this technique. On the other hand, object technology can be used with relatively small and medium size systems because of the difficulty of dealing with complexity and distribution using this technique.

2) The applications domain using agency technology is that which uses the distributed environment, e.g. communications systems, electronic commerce systems, air navigation systems, etc. As well as information systems that use data frequently, for example data mining systems, systems that retrieve data from the Internet, and so on. And systems that manage other branches such as network management, observe and control, and real-time systems. On the other hand, the applications domain using object technology is that of management systems that require medium-sized requirements, e.g. enterprise management systems, education, commercial, industrial, etc., additionally the design of web sites and web based applications.

The future work of the research is a comparison between the two techniques in a practical way, that is, a software is programmed using the two techniques, and then the results of the comparison are analyzed at each stage of the software evolution, highlighting and evaluation of the pros and cons at each stage. As well as evaluating and comparing their communication methods.

REFERENCES

[1] Vliet. H., "Software Engineering: Principles and Practice", John Wiley & Sons, Inc., 2nd edition, ISBN 0471975087, 2001.

[2] Booch, G., Rumbaugh, J., Jacobson, I., "Unified Modelling Language User's Guide", Addison-Wesley, 1999.

[3] Pressman, R., Maxim, R, "Software Engineering: A Practitioner's Approach", 8th Edition, McGraw-Hill, New York, USA, ISBN 978-0-07-802212 -8, 2015.

[4] Armstrong., J., "The Quarks of Object-Oriented development", Communications of the ACM, Vol. 49, No. 2, pp. 123-128, ISSN 0001-0782, 2006.

[5] Morris, M., Speier, C., Hoffer, G., "An Examination of Procedural and Object-Oriented Systems Analysis Methods: Does Prior Experience Help or Hinder Performance?" Decision Sciences, Vol. 30, No. 1, pp. 107-136, 1999.

[6] Vessey, I., Conger, S., "Requirements Specification: Learning Object, Process, and Data Methodologies", Communications of the ACM, Vol. 37, No. 5, pp. 102-113, ISSN 0001-0782, 1994.

[7] Glass, R. "The Naturalness of Object Orientation: Beating a Dead Horse?", IEEE Software, Vol. 19, No. 3, 104 p., ISSN 0740-7459, 2002.

[8] Jennings, N., Wooldridge, M., "Agent-Oriented Software Engineering", In F. J. Garijo and M. Boman, editors, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99), Vol. 1647, and pp.1-7, Springer-Verlag: Heidelberg, Germany, 1999.

[9] Jennings, N., "An Agent-Based Approach for Building Complex Software Systems", Communications of the ACM, Vol. 44, No. 4, pp. 35-41, ISSN 0001-0782, 2001.

[10] Odell, J., "Objects and Agents Compared", Journal of Object Technology, Vol. 1, No. 1, pp. 41-53, 2002.

[11] Adnan, G., Husaam, A., Mohamed, O., " Agent Vs Object with an in-depth insight to Multi-Agent Systems", IJARCS, Volume 4 , No. 8, May-June 2013.

[12] Luck M., "Agent-based Computing" GEOconnexion International Magazine, 2006.

[13] Uhrmacher A., Tyschler P, and Tyschler D., "Concepts of object-and agent-oriented simulation." Transactions of the Society for Computer Simulation 14.2: 59-67, 1997.

[14] Booch, G., "Object-Oriented Analysis and Design with Applications", 2nd Edition, Addison-Wesley, Reading, MA, USA, 1994.

[15] Jennings, N., Sycara, K., Wooldridge, M., "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems vol. 1, pp.275-306, 1998.

[16] IEEE Std 610.12., IEEE Standard Glossary of Software Engineering Terminology, 1990.

[17] Siau, K., Rossi, M., "Evaluation of Information Modeling Methods – A Review", in Proc. 31 Annual Hawaii International Conference on System Science, pp. 314-322, 1998.

[18] Priestley, M., "Practical Object-Oriented Design with UML", McGraw-Hill 2000.

[19] Pressman, R., "Software Engineering: A Practitioner's Approach", 5th Edition, McGraw-Hill, New York, USA, ISBN 0073655783, 2007.

[20] Tveit, A., "A survey of Agent-Oriented Software Engineering" First NTNU CSGSC, Computer Science Graduate Student Conference, Norwegian University of Science and Technology, Norway, 2001.

[21] Bradshaw, M., "An Introduction to Software Agents", In Jeffrey M. Bradshaw, editor, Software Agents, AAAI Press / The MIT Press, pp. 3-46, 1997.

[22] Yu, E., Liu, L., "Modeling Trust in the i* Strategic Actors Framework", Proceedings of the 3rd Workshop on Deception, Fraud and Trust in Agent Societies, Barcelona, Catalonia, Spain, 2000.

[23] Dardenne, A., Lamsweerde, A., Fickas, S., "Goal-Directed Requirements Acquisitions", Science of Computer Programming, Vol. 20, pp. 3-50, 1993.

[24] Jennings, N., "On Agent-based Software Engineering", Artificial Intelligence Vol. 117, pp. 277-296, 2000.

[25] Hadar, I., Leron, U., "How Intuitive is Object-Oriented Design?", Communications of the ACM, Vol. 51, No. 5, pp. 41-46, ISSN 0001-0782, 2008.

[26] Dam, H., K., “Supporting Software Evolution in Agent Systems", PhD thesis, RMIT University, Melbourne, Victoria, Australia, 2008.

[27] Sampson, J., "Raising the Level of Abstraction: On the Application of UML for Multiagent Systems Design", Jennifer Sampson DIF 8901 OOS, Trondheim, Norway, 2003.

[28] Wooldridge, M., Jennings, N., "Software Engineering with Agents: Pitfalls and Pratfalls", IEEE Internet Computing, Vol. 3, No. 3, pp. 20-27, 1999.

[29] Bergenti, F., Poggi, A., "Exploiting UML in the Design of Multi-Agent Systems", Proceedings ECOOP Workshop on Engineering Societies in the Agents' World 2000 (ESAW'00), Springer, pp. 96-103, 2000.

[30] Wooldridge, M., Jennings, N., "Intelligent Agents: Theory and Practice", The Knowledge Engineering Review, Vol. 10, No. 2, pp. 115–152, 1995.

[31] Scacchi, W., "Process Models in Software Engineering", In J.J. Marciniak (ed.), Encyclopedia of Software Engineering, 2001.

[32] Aylett, R., Brazier, F., Jennings, N., Luck, M., Nwana, H., Preist, C., "Agent Systems and Applications", Knowledge Engineering Review, Vol. 13, No. 3, pp. 303-308, 1998.

[33] Vidal, J., Buhler, P., Huhns, M., "Inside an agent", IEEE Internet Computing, Vol. 5, Issue 1, pp. 82-86, 2001.

AUTHOR BIOGRAPHY

Maher T. Alasaady was born in Mosul, Iraq, in 1980. He received the Diploma degree in computer systems from Technical Institute / Mosul, Mosul, Iraq, B.Sc. degree in software engineering from Mosul, University of Mosul, Iraq, in 2012, the M.Sc. degree in software engineering from Mosul, University of Mosul, Iraq. He is currently an Assistant Lecturer in the, Computer Systems department, Northern Technical University, Mosul, Iraq. He has authored three papers. His current research interests include software engineering, software agents, and database management.