# Evaluation of the Testing Methods in Agent-Oriented Software Engineering

**A. Saeed Zamani [1], B. Ramin Nassiri [2] and C. Sam Jabbehdari [1]**
[1]Department of computer engineering, Islamic Azad University, Tehran north branch, Iran
[2]Department of computer engineering, Islamic Azad University, Tehran central branch, Iran

**Abstract -** *Testing is an important process that can assure the quality and the correct functionality of the multi agent systems (MAS). Multiple testing methods in agent-oriented software engineering (AOSE) have been introduced in recent years. Although the quality of the system is dependent on the quality of the applied test method, very little attention has been paid to evaluating these testing methods. As a result, it is difficult to select a sufficient method for testing an agent-based system. Additionally, there are no means to determine what the advantages and drawbacks of each method are. This paper proposes a framework for evaluating and comparing the testing methods in AOSE. This framework addresses major divisions of a testing method. The framework is then used to evaluate some prominent testing methods, have been proposed so far. A subset of these testing methods, which cover more criteria in the proposed framework, is presented.*

**Keywords:** Agent-Oriented Software Engineering, Testing Methods, Evaluation Framework, Comparing the Test Methods.

## 1 Introduction

Agent-Oriented Software Engineering (AOSE), is concerned with how to specify, design, implement, verify (including testing and debugging), and maintain agent systems [36]. The objective of AOSE is to efficiently and effectively develop high-quality agent-based software products. Nowadays, intelligent agent-based systems are applied to many domains including robotics, network security, traffic control, and ecommerce. Therefore, the owners and the operators of these systems need guarantees over quality and correct functionality of them [11]. This calls for suitable software engineering frameworks, including testing techniques, to provide high-quality software development processes and products [23].

During the last decade, many methodologies have been proposed for developing agent-based systems but current state of AOSE paradigm reports relative lack of industrial acceptance [1]. Furthermore, the application of these methodologies is still limited due to their lack of maturity and standardization. Testing is one of the most urgent activities that are often disregarded in most agent-oriented methodologies [7]; mainly because they focus on analysis and design activities, and relegate the implementation and testing to the traditional techniques [20]. Software testing is one of the most important phases in software engineering, and plays a pivotal role in software quality assurance.

Under ideal situations, with minimal testing efforts, integration of reliable software agents should produce high-quality agent-based systems. In reality, however, many agent-based software characteristics, such as autonomy, pro-activity, mutual relationships of these agents and relationships with the environment impose great difficulties on achieving this goal and make traditional techniques inefficient.

To thoroughly understand the difficulties and key issues in testing and maintaining the agent-based software, and thereby to apply adequate methods, this paper focuses on the following questions:

1. What are the key characteristics of agent-based systems that distinguish them from other systems (merely according to testing)?
2. How can agent oriented software testing methods verify these characteristics?

In order to answer these questions, the testing issues are characterized by proposing a framework to evaluate the existing testing methods. We consider the methods employed by agent-oriented methodologies (there are also several methodologies that do not include testing in their process models [7]) and the methods that are not related to any specific methodology (e.g. [9]). Comparing prominent agent-oriented testing methods and evaluating their strengths and weaknesses, play an important role in improving their performance. This can also contribute to applying appropriate testing methods or combinations of various methods and techniques. Within the last few years many frameworks have been proposed for evaluating AOSE methodologies, e.g. [17], [35]. These frameworks merely check if the testing process is mentioned in the methodologies or not. There is no work on verifying the quality of the testing methods in AOSE methodologies. Yet, comparing methods is often difficult, because they might address different aspects or differ in their terminology. For instance, some methods verify the static structure of the agent systems [4], [23], while the others verify the dynamic behavior [9]; some focus on the *Agent Level* of the systems while some consider that the agents are reliable and focus on the *Society Level* of the

systems. Comparing is also problematic with some methods that are influenced by a specific methodology (e.g. *Tropos*, *MaSE* and *Prometheus*).

This paper is organized as follows. Section 2 describes our proposed framework for comparing and evaluating AOSE testing methods. In Section 3 the framework is applied in order to compare the existing multi-agent testing methods and finally Section 4 concludes the paper.

# 2 The Evaluation Framework

In [11], six testing method were evaluated. The evaluation criteria are divided into two test levels and their test types (*white box* and *black box*). In this paper, a comprehensive framework of evaluating and comparing agent-oriented testing methods is proposed. This framework offers a well-defined, structured set of aspects that an agent-oriented testing method should include. The first major division of the framework is based on the framework suggested by [2]. This study extends and modifies this framework to address the properties of a comprehensive testing method in AOSE. The other major divisions that are being inspired by [3] and [35] are not specifically related to AOSE and could be considered in other software engineering paradigms, e.g. *object oriented* and *procedural*. We refer to a testing method as the entire set of these major divisions:

- Multi-agent systems test basics
- Test process
- Test techniques
- Test pragmatism

Each of these four major divisions includes their specific criteria that will be explained in the following. The proposed framework is illustrated in figure 1.

We emphasize that these four divisions, are in fact four different views of the whole test method, and can overlap; i.e. some of the criteria may be present in different divisions, having different names but the same identity.

## 2.1 Multi Agent Systems Test Basics

Multi-agent systems testing are normally divided into multi layers [9], [11] and [29]. According to the *V-model* [31], the *Test Level*s are: testing agents as individual units of the MAS, testing the integration of collaborator agents and testing the whole system. *Dynamic Testing* (will be discussed in 2.3) should be used in the first layer, in order to verify the behavior of an agent. Much of this testing would require another agent to trigger an event inside the agent to be tested, such as a message from another agent, or an event from the environment [9]. Furthermore, *Static Testing* (will be discussed in 2.3) should be used for validation in the first layer.

We have changed the general V-model by adding a layer called "*Agent Acceptance Test*" after testing the functionality of an agent in the first layer. This test layer is concerned with the essential properties of agents such as *Autonomy*, *Pro-activity* and *Sociability*. Testing agents, according to these properties, is the most challenging task which makes the traditional testing methods insufficient.

Two main issues in *Integration Level* testing of an agent-based system are to be considered: (i) data models define the contents and format of the interactions in the control protocol and (ii) as agent-based systems are built under a distributed environment, which will then inherit all issues of the distributed systems, such as race conditions and deadlocks.

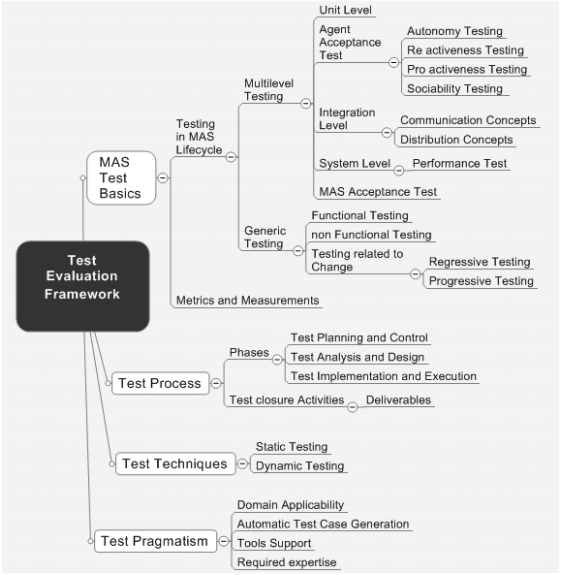


Figure 1 - Multi Agent Systems Test Evaluation Framework.

Agents are different from objects and interaction is based on communication language and protocols, rather than invoking the functions of each other. In addition, the agent-based systems include agents that *autonomously* pursue their individual goals and access resources and services of the environment. Therefore, deadlock detection techniques must differ from other distributed systems. A set of errors that could occur in unit level and integration level of agent-based system are presented in [28].

After the end of the *System Level* testing, the functionality of the whole system could be assured. Furthermore, *Performance Testing* is needed to verify that all of the worst case performance targets have been met (according to the resource constraints within the system, e.g., time, CPU and memory). Since these systems are also non-deterministic [11], this kind of testing will be challenging too. Within the last layer, *Validation* should be performed to find out whether the system has met the stakeholders requirements or not.

*Testing types* (e.g. *functional*, *nonfunctional* and *regression*) are independent of a particular test level. In

the *Generic Testing* division, the evaluation is about the existence of test types in the test levels. For instance, since agents are work flexibly in a dynamic environment without continuous direct supervision [11], and may change through the time, *regression testing* (for parts that have been changed) and *progressive testing* (for parts that have been added) must be performed in the *integration level*.

The last important factor in testing is how to define *Quality Metrics* in AOSE. Well-established metrics and measures, aligned with project objectives, will enable the tester to track and report the test and quality results. A lack of metrics and measurements leads to subjective assessments of quality and testing [3]. Not only metrics and measurements are crucial, but also *baselines* (An acceptable result), are required to verify the actual quality of the system under test, against the expected quality. Although these metrics are different from traditional software metrics, only few studies have addressed the issues of AOSE metrics.

## 2.2 Test Process

We should investigate the way that any testing method looks at the *Test Process* in the AOSE. If the method does not consist of phases then it more looks like an activity, rather than a process, and may delay testing until the end of implementation. According to *ISTQB* framework[1] [3], a test process consists of the following activities:

- Planning and control
- Analysis and design
- Implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

For any particular testing method, *Test Process criteria* involve clarifying what activities of a software testing process are mentioned within the agent-based system lifecycle.

*Test Planning* sets a framework for deriving *Test Cases*[2] and *Test Conditions* from the *Test Basis*. The test basis may include *requirements specifications*, *design specifications*, *quality risks*, and some other items. In the *Test Control*, the test method compares actual progress against the plan. The *Test Objectives* are a major deliverable. The *Test Analysis and Design* involves the following sub-activities:

- Identifying and refining the test conditions for each test objective.
- Creating test cases that exercise the identified test conditions.
- Creating *Test Oracles* (will be discussed in 2.3).

*Test Implementation* includes all the remaining tasks necessary to execute the test cases. In this activity, the test method should run a Single Test Procedure and log the Test Results. The *Evaluation of Exit Criteria* and reporting of results is a test management activity. Delivering test work products (e.g. error reports, test plan, etc.) is one of the *Test Closure Activities* that, a test method could have.

## 2.3 Test Techniques

There are two kinds of *Testing Techniques* that presented in the first level of the test techniques division [3]: (i) static testing (i.e. testing the system without running it) and (ii) dynamic testing (i.e. testing the system during its runtime). The test techniques are applied in the test types (presented in 2.1).

A sufficient testing method should cover both the static and the dynamic testing techniques. The input of the static testing could be the *AOSE Artifacts* that get developed in the agent-oriented methodologies [11]. Static and dynamic testing inputs are illustrated in figure 2. The *Model checking* approaches seem to be more acceptable static techniques, (because of having less complexity and better traceability [19]), since these methods propose that testing could be in some way based on the models of the system, which are abstractions of the actual system, and can be used for automated generation of test cases. Static testing has also the potential to lead to more accurate requirements *Verification*.

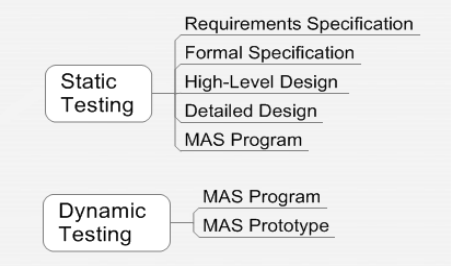


Figure 2- Input Artifacts for Static and Dynamic Testing.

On the other hand, dynamic testing is needed for validating the behavior of agents and the MAS as a whole. White box testing and black box testing are two common dynamic testing. White box testing can be performed in a traditional way, while there are some problems with the black box testing: It is very hard to find a test oracle (i.e. a source to determine expected results to compare with the actual result of the software under test [12]) for black box testing because of self-adaptation, learning and the autonomy of agents and successive tests with the same test data may give different results [28].

## 2.4 Test Pragmatism

In this division, we examine the practical aspects of using a test method within an agent oriented methodology. If the method is independent from a

1 International Software Testing Quality Board

2 The comprehensive definitions of the aforementioned testing terms can be found in [12].

particular methodology, then it becomes more applicable. Some methods that are evaluated in section 3 are proposed for specific agent architectures and agent-oriented methodologies. On the other hand, the methods that are proposed within AOSE methodologies seem more reliable. The main reason of the limited applicability of testing methods is that they are very challenging and expensive since it is quite complicated to automate them [6].

A sufficient testing method should propose a supporting tool and an automatic test case generator to reduce the time required for testing [27], and have *visualization* techniques, to become more acceptable.

These methods could also be evaluated on the mathematical sophistication level (e.g. exploiting the *Petri Nets*) and knowledge (e.g. *BDI* architecture and *Formal* methods) required to fully exploit the method. This consideration could enable an AOSE methodology to adopt a test method, better within its process.

## 2.5 Metric

In order to rank the properties examined in the evaluation process, we propose a scale of 1 to 3 as follows:

*1*: Indicates that the test method does not address the property.

*2*: Indicates that the test method refers to the property but not enough details are provided.

*3*: Indicates that the method addresses the property with a particular technique.

We emphasize that these numbers are not representing the quality of each property. They only indicate the existence of a property in the relevant methods; albeit with an exception for the *Test Pragmatism* that will be explained in section 3.

Table 1 – The Evaluation of the Test Methods in AOSE

| The Evaluation of Test Methods in AOSE | Test Criteria | MAS Test Basics | | | | | | | | | | | | | | Test Process | | | | Test Techniques | | Test Pragmatism | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Test Methods | | Unit | Autonomy | Reactivity | Pro-activity | Sociability | Communication | Distribution | System | Performance | Acceptance | Metrics & Measurement | Functional | Non Functional | Change Related | Planning & Control | Analysis & Design | Implementation | Closure Activities | Static | Dynamic | Domain Applicability | Tool Support | Required Expertise | Automatic Test Case generation |
| [29] 3-layer | | 3 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 1 | 3 | 3 | 1 | 1 | 2 | 2 | 1 | 1 | 3 | 3 | 1 | 1 | 1 |
| [18] Automated BDI | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 3 | 1 | 1 | 2 | 3 | 3 | 1 | 3 | 1 | 1 | 3 | 3 | 3 |
| [32,33] Conversation Verification | | 3 | 1 | 1 | 1 | 1 | 3 | 3 | 1 | 1 | 1 | 1 | 3 | 3 | 1 | 1 | 3 | 3 | 2 | 3 | 3 | 1 | 3 | 3 | 3 |
| [27] Design artifacts | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 2 | 1 | 3 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| [22] Evolutionary Testing | | 2 | 2 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 1 | 3 | 3 | 1 | 2 | 3 | 3 | 3 | 3 | 1 | 3 | 3 | 3 | 1 | 3 |
| [23] Goal-Oriented | | 3 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 2 | 3 | 2 | 2 |
| [13] INGENIAS | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 2 | 1 | 1 | 3 | 2 | 1 | 2 | 2 | 2 | 1 | 3 | 3 | 1 | 3 | 1 | 1 |
| [8] JAT | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 2 | 3 | 3 | 2 | 1 | 3 | 3 | 3 | 1 | 2 |
| [15] MadKit | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 3 | 1 | 1 | 3 | 3 | 3 | 1 | 1 | 3 | 1 | 1 | 3 | 3 | 3 | 1 | 1 |
| [6] MAZBD | | 3 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 1 | 3 | 1 | 2 | 2 | 2 | 3 | 1 | 1 | 3 | 2 | 3 | 1 | 3 |
| [9] Mock Agent | | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 2 | 2 | 3 | 2 | 1 | 3 | 3 | 3 | 1 | 2 |
| [19] Model-based Deadlock detection | | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 3 | 3 | 2 | 3 | 3 | 2 | 3 | 2 | 2 |
| [24,25] Ontology-Based | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 2 | 3 | 2 | 1 | 2 | 3 | 3 | 1 | 1 | 3 | 3 | 3 | 1 | 3 |
| [26] Prometheus | | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 1 | 1 | 3 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 3 | 1 | 3 | 3 | 1 |
| [30] Regression Testing | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 2 | 1 | 1 | 1 | 3 | 3 | 1 | 1 | 1 |
| [10] SEAUnit | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 2 | 2 | 1 | 3 | 3 | 2 | 1 | 1 | 2 | 1 | 3 | 3 | 3 | 3 | 1 | 1 |
| [34] SUNIT | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 2 | 1 | 3 | 2 | 2 | 2 | 2 | 3 | 2 | 3 | 3 | 2 | 3 | 2 | 3 |
| [28] Test Agent | | 3 | 1 | 1 | 1 | 1 | 3 | 3 | 2 | 1 | 1 | 3 | 3 | 3 | 3 | 1 | 2 | 3 | 1 | 3 | 3 | 3 | 2 | 1 | 2 |
| [4] Verifying by model checking | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 3 | 2 | 1 | 1 | 1 | 3 | 2 | 3 | 1 | 2 | 3 | 2 | 1 |
| [16] XP | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 2 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 3 | 1 | 2 | 1 | 2 |
| [21] Zeus | | 3 | 1 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 1 | 3 | 3 | 2 | 2 | 2 | 1 | 3 | 3 | 3 | 3 | 2 | 3 | 1 | 1 |

# 3 The Evaluation of Testing Methods in AOSE

In this section we evaluate the selected testing methods found in literature for the AOSE approach, according to the framework presented in Section 2. The evaluation and the points that each method has gained, are presented in Table 1.

In the first division of the framework, we evaluate that which testing layers are covered in the method. Then, the test types employed in each test layer are checked, and finally we verify the existence of any metrics and measurements proposed in the method. We rank the MAS test basics of each method (discussed in 2.1), according to the metrics presented in 2.5.

For the purpose of ranking the test process criteria, we emphasize that whenever a method does not explicitly mention the testing activities (discussed in 2.2), we rank it based on a personal analysis that may not necessarily reflect the original intentions of the proposers, and that sometimes has to sharpen shades.

Whether a method performs during the development phase without running the system or not, we simply gave the static testing criterion the ranks of 3 and 1, respectively. We use the same ranking scheme for the dynamic testing criterion, to indicate if a method performs on the running agent-based system.

Based on the tool support, automatic test case generation, visualization and knowledge level that may be suggested by a method, we evaluate the pragmatics aspect of a method (discussed in 2.4). Furthermore, we investigate whether the method proposed particular domain of applicability or development methodology to use the method, or we can use the method within variant methodologies. The only exception here is that the given points indicate the quality of each property in addition to its existence.

Due to lack of space we will not discuss the justification of each given point to a relevant method, which we leave to future work for the selected methods. As illustrated in Table 1, there is no single best method to achieve the highest score in all criteria. Therefore, in order to choose a comprehensive method for testing in AOSE, that includes sufficient essential properties, we have to combine several existing methods.

As it is not possible to join all methods, we need to find the smallest subset of all *interesting* methods. Interesting are all methods that are not worse than any other method in all criteria. We use an approach called *Skyline* [5] to find the subset of interesting methods. Table 2 illustrates the skyline of the evaluated methods, ordered by the total points that each method has gained. Any method left out of this skyline is dominated by at least one method presented in the skyline and can be disregarded in the combination of methods.

The size of the skyline is still large. It shows that most of the proposed testing methods in AOSE, present an approach suitable for at least one group of the MAS developers. Furthermore, the large size of the skyline emphasizes the lack of a comprehensive testing method in AOSE. From the Skyline, we can now make our final decision, thereby weighing our personal preferences for testing criteria.

Table 2 – The Skyline of the Test Methods in AOSE

| The Test Method | Total Point |
|---|---|
| [23] Goal-Oriented | 57 |
| [27] Design artifacts | 52 |
| [22] Evolutionary Testing | 51 |
| [34] SUNIT | 50 |
| [28] Test Agent | 50 |
| [32,33] Conversation Verification | 49 |
| [21] Zeus | 48 |
| [15] MadKit | 46 |
| [10] SEAUnit | 46 |
| [24,25] Ontology-Based | 45 |
| [19] Model-based DL detection | 44 |
| [13] INGENIAS | 43 |
| [29] 3-layer | 40 |

# 4 Conclusion

In this paper, we investigate the essential aspects that an agent-oriented test method should include. The proposed framework in section 2 divides these aspects into the four major divisions: MASs test basics, test process, test techniques and pragmatism. All these criteria are explained in section 2. Section 3 demonstrates the use of the proposed framework by performing an evaluation of some prominent testing methods. We conclude that there is no method that covers all criteria. As a result, in order to find a sufficient test method we have to combine several different methods. The selected subset of testing methods is presented in Table 2. The combination of these methods fulfills most considered criteria. However, there are some criteria which are disregarded by most of the methods, e.g. the *agent acceptance testing*. Furthermore, the evaluation demonstrates low points in the *performance test*, the *metrics and measurements* and the *test process*. In our future work, we plan to devise a method to promote the testing process in AOSE, by dominating the uncovered criteria and performing a standard testing process, proposed by ISTQB.

# 5 References

[1] Akbari, Z. “A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance”; Journal of Computer Engineering Research, Vol. 1, Issue 2, pp. 14-28, April 2010.

[2] Ayatollahzadeh Shirazi, M.R., Abdollahzadeh Barfouroush, A. “A Framework for Agent-Oriented

Software Engineering Based On an Analytical Survey"; Iranian Journal of Electrical and Computer engineering, Vol. 6, Issue 1, pp. 36-47, 2007.

[3] Black, R. "Guide to the ISTQB Advanced Certification as an Advanced Test Manager". Rocky Nook, Vol. 1, 2009.

[4] Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M. "Verifying multi-agent programs by model checking"; Autonomous Agents and Multi-Agent Systems, Vol. 12, Issue 2, pp. 239-256, 2006.

[5] Borzsonyi, S., Kossmann, D., and Stocker, K. "The skyline operator"; In Proceedings of the International Conference on Data Engineering (ICDE'01), pp. 421-430, 2001.

[6] Caire, G., Cossentino, M., Negri, A., Poggi, A., and Turci, P. "Multi-agent systems implementation and testing"; From Agent Theory to Agent Implementation - Fourth International Symposium (AT2AI-4), Vienna, Austria, April 2004.

[7] Cernuzzi, L., Cossentino, M., Zambonelli, F. "Process Models for Agent-based Development"; Journal of Engineering Applications of Artificial Intelligence, Vol. 18, Issue 2, pp. 205-222, 2005.

[8] Coelho, R., Cirilo, E., Kulesza, U., von Staa, A., Rashid, A., Lucena, C. "Jat: A test automation framework for multi-agent systems"; In Proceedings 23rd IEEE International Conference on Software Maintenance (ICSM07), pp. 425-434, 2007.

[9] Coelho, R., Kulesza, U., von Staa, A., Lucena, C. "Unit Testing in Multi-Agent Systems using Mock Agents and Aspects"; In Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, 83-90, 2006.

[10] Ekinci, E.E., Tiryaki, A.M., Çetin, Ö. "Goal-oriented agent testing revisited"; In proceedings of the Ninth International Workshop on Agent-Oriented Software Engineering, pp. 85-96, 2008.

[11] Gatti, M.A.C., Staa, A.V. "Testing & debugging multi-agent systems: a state of the art report"; Departamento de Informa tica, PUC-Rio, Rio de Janeiro, 2006.

[12] Glossary Working Party. "Standard glossary of terms used in Software Testing"; International Software Testing Qualifications Board, 2010.

[13] Gomez-Sanz, J.J., Botía, J., Serrano, E., Pavón, J. "Testing and debugging of MAS interactions with INGENIAS"; In proceedings of the Ninth International Workshop on Agent-Oriented Software Engineering, pp. 133-144, 2008.

[14] Graham, D., Van Veendendal, E., Evans, I., Black, R.; "Foundations of Software Testing ISTQB Certification". Patrick Bond, 2008.

[15] Huget, M.P., Demazeau, Y. "Evaluating multiagent systems a record/replay approach"; In Proceedings of the IEEE/WIC/ACM International Conference of Intelligent Agent Technology (IAT 2004), pp. 536-539, 2004.

[16] Knublauch, H. "Extreme programming of multi-agent systems"; In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 704-711 2002.

[17] Lin, C.E., Kavi, K.M., Sheldon, F.T., Daley, K.M., Abercrombie, K. "A Methodology to Evaluate Agent Oriented Software Engineering Techniques"; In Proceedings of the 40th Hawaii International Conference on System Sciences, pp. 1-10, 2007.

[18] Low, C.K., Chen, T.Y., Rönnquist, R. "Automated Test Case Generation for BDI agents"; Autonomous Agents and Multi-Agent Systems, Vol. 2, Issue 4, pp. 311-332, 1999.

[19] Mani, N., Garousi, V., Far, B.H. "Testing Multi-Agent Systems for Deadlock Detection Based on UML Models"; In proceedings of the 14th International Conference on Distributed Multimedia Systems (DMS08), Boston, USA, pp. 77-84, 2008.

[20] Moreno, M., Pavon, J., Rosete, A. "Testing in Agent Oriented Methodologies"; Omatu et al. (Eds.): IWANN, Part II, LNCS 5518, Springer-Verlag Berlin Heidelberg, pp. 138-145, 2009.

[21] Ndumu, D.T., Nwana, H.S., Lee, L.C., Collis, J.C. "Visualization and debugging distributed multi-agent systems"; In Proceedings of the third annual conference on Autonomous Agents (ACM press), pp. 326-333, 1999.

[22] Nguyen, C.D., Perini, A., Tonella, P. "Constraint-based Evolutionary Testing of Autonomous Distributed Systems"; In proceedings of IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08), pp. 221-230 2008.

[23] Nguyen, C.D., Perini, A., Tonella, P. "A goal-oriented software testing methodology"; Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, Springer, Heidelberg, vol. 4951, pp. 58-72, 2008.

[24] Nguyen, C.D., Perini, A., Tonella, P. "Experimental Evaluation of Ontology-Based Test Generation for Multi-agent Systems"; M. Luck J.J., Gomez-Sanz (eds.): Agent-Oriented Software Engineering, Springer-Verlag Berlin Heidelberg, pp. 187-198, 2009.

[25] Nguyen, C.D., Perini, A., Tonella, P. "Ontology-based test generation for multi agent systems (short paper)"; In Proceedings of the 7th International

Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008). Estoril, Portugal, pp. 12-16, 2008.

[26] Padgham, L., Winikoff, M., Poutakidis, D. “Adding Debugging Support to the Prometheus Methodology”; Engineering Applications of Artificial Intelligence, special issue on Agent-oriented Software Development, Volume 18, Issue 2, pp. 173-190, March 2005.

[27] Poutakidis, D., Winikoff, M., Padgham, L., Zhang, Z. “Debugging and Testing of Multi-Agent Systems using Design Artefacts”; R.H. Bordini et al. (eds.), Multi-Agent Programming, Springer Science + Business Media, pp. 215-258, 2009.

[28] Rouff, C. “A Test Agent for Testing Agents and Their Communities”; Aerospace Conference Proceedings, IEEE Volume 5, pp. 2633-2638, 2002.

[29] Salamon, T. “A Three-Layer Approach to Testing of Multi-agent Systems”; G.A. Papadopoulos et al. (eds.): Information Systems Development, DOI 10.1007/b137171_41, Springer ScienceBusiness Media, pp. 393-401, 2009.

[30] Srivastava, P., R., Anand, K., Reddy, S., Raghurama G. “Regression Testing Techniques for Agent Oriented Software”; In Proceedings of the International Conference on Information Technology, pp.221-225, 2008.

[31] The V-Model: The Development Standards for IT Systems of the Federal Republic of Germany 2005, *http://www.v-modell.iabg.de/* (cited February 2011).

[32] Timothy, H.L., DeLoach, S.A. “Automatic Verification of Multiagent Conversations”; In the Annual Midwest Artificial Intelligence and Cognitive Science Fayetteville, 2000.

[33] Timothy, H.L., DeLoach, S.A. “Verification of Agent Behavioral Models”; In Proceedings of the International Conference on Artificial Intelligence (IC-AI'2000), 2000.

[34] Tiryaki, A.M., Oztuna, S., Dikenelli, O., Erdur, R.C. “Sunit: A unit testing framework for test driven development of multi-agent systems”; Padgham, L., Zambonelli, F. (eds.): AOSE VII / AOSE 2006. LNCS, Springer, Heidelberg, vol. 4405, pp. 156-173 2007.

[35] Sturm A., Shehory, O. “A Framework for evaluating agent-oriented methodologies”; In Proceedings of Workshop on Agent-Oriented Information System (AOIS), Melbourne, Australia, pp. 60-67, 2003.

[36] Winikoff, M. “Future Directions for Agent-Based Software Engineering”; Special section on Future of software engineering and multi-agent systems, International Journal of Agent-Oriented Software Engineering (IJAOSE08), pp. 1-10, 2008.