The Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

University of LARBI BEN M'HIDI

-Oum El Bouaghi-

Faculty of Exact Sciences and Natural and Life Sciences

Department of Mathematics and Informatics

Research Laboratory On Computer Science's Complex Systems

Number of order: ……..….

Series: ……..

Multi Agent Systems And Their Testing Efforts: An Empirical Approach

A dissertation submitted in partial fulfilment for the degree of:

DOCTORATE OF SCIENCE in Computer Science

(Option: Artificial Intelligence & Imagery)

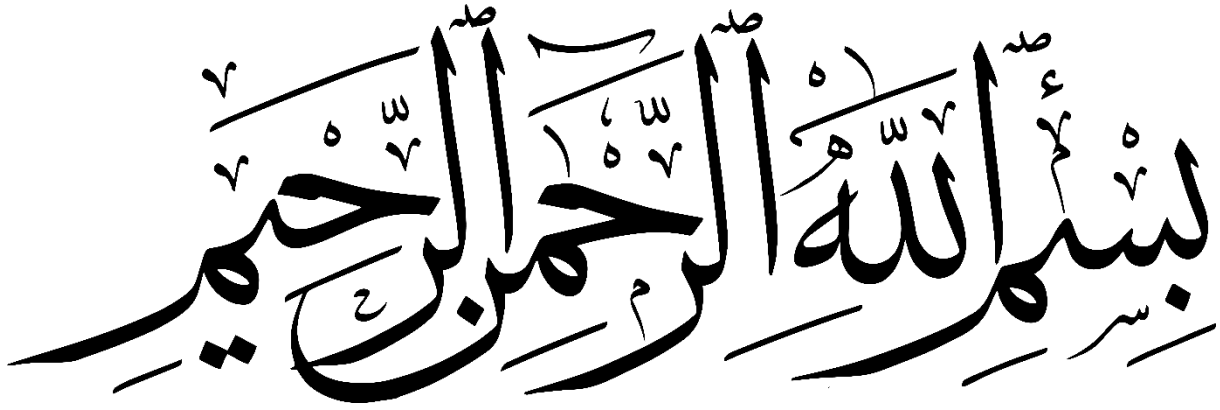
By:

Ayyoub KALACHE

Doctoral Committee:

Dr. BENABOUD RohAllah	Chair	University of Oum El Bouaghi.
Prof. MOKHATI Farid	Supervisor	University of Oum El Bouaghi.
Prof. BABAHENINI M.Chaouki	Co-Supervisor	University of Biskra.
Dr. MARIR Toufik	Examiner	University of Oum El Bouaghi.
Prof. HOUASSI Hichem	Examiner	University of Khenchela.
Dr. SOUIDI Mohammed El Habib	Examiner	University of Khenchela.

2025


بسم الله الرحمن الرحيم

Multi Agent Systems and Their Testing Efforts:

An Empirical Approach

All praise is due to Allah,

by whose favor good deeds are accomplished.

To my parents Abd el Baki and Chebout Ardjouna,

who made me the man I am today.

To my big brother, teacher, and supervisor Farid Mokhati,

who shaped me into the Software developer I am today.

To my mentor and supervisor Mourad Badri,

who guided me to become the researcher I am today.

To my siblings Nassima, Abd-el-Ali, Abd-el-Malek, and Mahmoud,

Sorry for being absent over the last decade...

To my wife Sabrine,

for your unwavering support and patience.

To my daughter Alicia,

who stretched it two years longer.

To myself

For not withdrawing.

Acknowledgement

All praise and gratitude are due to Almighty Allah, whose infinite mercy, guidance, and blessings have been the foundation of every success in my life." *Glory be to You! We have no knowledge except what You have taught us. You are truly the All-Knowing, All-Wise*" (Surah Al-Baqarah - 32).

I would like to express my deepest gratitude to my supervisors, Prof. Farid Mokhati and Prof. Mourad Badri, whose expertise, guidance, and support were invaluable throughout the course of this research. Your patience, encouragement, and insightful feedback helped me navigate the complexities of this work.

I would like to extend my thanks to my supervisor Prof Mohamed Chaouki Babahenini, for his academic support and guidance.

I would like to stretch my heartfelt thanks to the jury members who honored me by reviewing this work and offering their constructive criticism and valuable remarks. My profound gratitude also goes to all my teachers for imparting their invaluable knowledge throughout my academic journey.

A special thank you to my wife for her unwavering support, patience, and for tirelessly proofreading my theses and papers.

I deeply appreciate the friendship and support provided by many during this project. Special thanks to my dear friends Cmdr. Messai Moustapha, Dr Salim Zeroughi, Dr Mohamed Chebout, and, whose companionship provided moments of comfort and distraction when I needed them most.

Finally, expressing gratitude to my parents, first teachers in my life, feels redundant, as mere words cannot capture their boundless affection. To convey my profound appreciation, I can only turn to:" *My Lord! Be merciful to them as they raised me when I was young.*" (Surah al-isra – 24).

Ayyoub Kalache

Abstract

Software agent-based technology is a key approach to addressing the complex, distributed, and autonomous required nature of modern software systems. However, quality assurance remains a significant challenge that hinders its industry adoption. This research investigates the widely held assumption that agent-based systems are inherently hard to test by empirically studying their testability from the perspective of agent test writing effort. Focusing on JADE-based multi agent systems (MAS), the properties, attributes, and testing efforts of a sample of JADE agents are characterised and statistically analysed using correlation and regression techniques. This analysis allowed to identify and understand the relationship between agent attributes and properties, and testing effort. These insights are then leveraged to build agent testing effort prediction models using machine learning techniques. The study addressed key challenges including the absence of comprehensive agent measurement models, the lack of a thorough JADE testing framework, and the unavailability of open MAS testing cases that could serve as case studies. Overall, this research resulted in several significant contributions to the field of agent based system quality, particularly in testing and measurement. It enhanced the understanding of agent testability and the way agent characteristics and properties affect it at different testing levels. It proposes a high effectiveness prediction models. It introduces the Unified Framework for Agent Properties Measurement (UFAPM), to tailoring a context-specific agent measurement model. It proposes JADE Testing Framework (JTF), a comprehensive and effective solution for JADE agent testing, sustained with over 4,000 lines of agent test code that provide a valuable benchmark for future studies.

Key Words

Multi-agent system, Testability, Test, Testing effort, JADE, Measurement, Properties, Attributes Machine learning, Correlation

الملخص

تعد تكنولوجيا برمجيات الوكلاء طريقة ناجعة لمعالجة الطبيعة المعقدة والموزعة والمستقلة المطلوبة للأنظمة البرمجيات الحديثة. ومع ذلك، يظل ضمان الجودة تحديًا كبيرًا يعيق اعتماد هذه التكنولوجيا في الصناعة. ينظر هذا البحث في الافتراض الشائع بأن الأنظمة القائمة على الوكلاء يصعب اختبارها بطبيعتها، وذلك من خلال دراسة قابليتها للاختبار من منظور جهد اللازم لاختبار الوكلاء.لقد تم اختيارمجموعة من الأنظمة متعددة الوكلاء القائمة على منصت JADE كعينة للدراسة، اين تم قياس لكل وكيل من العينة كل من خصائصه ، قدراته و كذا الجهد اللازم لاختباره .البيانات المتحصل عليها تم تحليلها إحصائيًا باستخدام تقنيات الارتباط والانحدار. مما سمح بتحديد وفهم العلاقة بين كل من خصائص و قدرات من جهة وكذا جهود الاختبار من جهة اخرى . ثم تم استخدام هذه النتائج لبناء نماذج تنبؤ لجهد اختبار الوكلاء باستخدام تقنيات التعلم الآلي . واجهت الدراسة التحديات عديدة شمل غياب نماذج كاملة و موثوقة لقياس خصائص وقدرات الوكلاء، عدم وجود منصة متكاملة لاختبارهم، غياب اختبارات مفتوحة المصدر و التي يمكن استخدامها كعينات لدراسة. بشكل عام، نتج عن هذه الدراسة عدة مساهمات هامة في مجال جودة الأنظمة القائمة على الوكلاء، خاصة في مجالي الاختبار والقياس حيث عززة فهم قابلية اختبار للوكلاء وكيفية تأثير خصائصهم و قدراتهم على الجهد اللازم لاختبارهم، تقترح نماذج تنبؤ ذات فعالية عالية ، إطارموحد خاص بصياغة نمادج قياس متكاملة للوكلاء(UFAPM) و كذا منصة اختبار متكاملة وفعالة (JTF) للأنظمة متعددة الوكلاء القائمة على JADE ، مدعوم بأكثر من 4,000 سطر من كود اختبار لوكلاء والذي يعتبر قاعدة بيانات قيمّة للدراسات المستقبلية .

الكلمات المفتاحية

نظام الوكلاء المتعدد، القابلية للاختبار، الاختبار، جهد الاختبار، JADE، القياس، الخصائص، قدرات ، التعلم الآلي، الارتباط.

Table of Contents

Table of Figures

Table of Tables

General Introduction

1 | Introduction

Software agent technology is one of the key approaches to dealing with nowadays complex, distributed and autonomous software systems requirements. They can evolve dynamically over time, to accommodate new components and meet new requirements (Cossentino et al., 2014). Despite the progress made in the field of agent-oriented software engineering (AOSE) since its inception in the ’80s, quality assurance of developed agent-based systems was and is still an open question (Mascardi et al., 2019; Winikoff, 2012; Zambonelli & Omicini, 2004; Wille et al., 2002). Winikoff (Winikoff, 2012) stated that “ *One of the strengths of agents and multi-agent systems is that they are able to deal with a range of situations, balancing proactivity and reactivity as needed… However, how can I be confident that my system will work –reasonably- in all situations ?*”.

Quality assurance via testing is one of the crucial phases in the development cycle of software and agent-based software make no exception. Yet the distinct nature of the later software type, that is, built to be autonomous, distributed, open, and deliberative systems; has made it gain the reputation to be inherently hard to test, and sub-sequentially to assure their quality (Ferrando & Malvone, 2022; Dix et al., 2021; Winikoff, 2017; Winikoff & Padgham, 2013). For instance Munroe et al. (2006) Stated *“…validation through extensive tests was mandatory . . . However, the task proved challenging for several reasons. First, agent-based systems explore realms of behaviour outside people’s expectations and often yield surprises*...”. This raises deep questions like:

- What makes a MAS in general and agent in particular hard to test?
- Which of the agent’s properties and characteristics influence its testability?

Surprisingly, there has been little work in the specialised literature (Gonçalves et al., 2019; Winikoff, 2017; Winikoff & Cranefield, 2014) on this matter, even though testing is one of the most used techniques for agent-based software verification (Bakar & Selamat, 2018). And, well understanding of agent-based system testability would not only increase the understanding of agent-based software quality in general (testability is a pertinent attribute of maintainability characteristic in the ISO 25010 software quality model (ISO, 2011)), but it would also make testing activity more efficient: better resource allocation and better test prioritisation (Terragni et al., 2020; Bruntink & Deursen, 2006).

Motivated by the aim to understand the testability of multi-agent systems in general and agents in particular, and the fact that this question is shallowly investigated in the literature, this D.Sc thesis presents a deep empirical investigation of Multi-Agent Systems (MAS) testability. The testability is examined from the perspective of agent testing effort, that is the effort required to write agent tests at both **unit** (units that make up the agent, for example: beliefs, inner behaviours, etc.) and **agent** (interaction) testing levels. It seeks to answer the following main questions:

- *What agent's property influences its testability, and on which testing level (unit or agent) does it do?*
- *What agent's attribute (capability) influences its testability and on which testing level (unit or agent) does it do?*
- *Could agent testing effort be predicted from its source code?*

The study has as its subject a Java Agent DEvelopment framework (JADE) agent. JADE platform was chosen, rather than other ones because it was and still is one of the most used platforms for agent-based software systems development (Wrona et al., 2023; Bergenti et al., 2020; Kravari & Bassiliades, 2015), and more essentially JADE is agent-architecture independent which leans to make the obtained results more generalised.

The study is conducted on the agent abstraction level only, for now, and not multi-agent as a whole system, since the agent is the building block of any MAS and any study of MAS testability can be done without initially well grasping and understanding this first abstraction level; furthermore, the study needs to be conducted and delivered with a D.Sc. research timeframe extending the study to system level is not feasible.

The main research idea is, that on a sample of JADE agents, agents' properties, attributes and testing efforts are first characterised and quantified. Then the collected data are statistically analysed to understand the relationship between these entities (properties/attributes Vs testability) and to build an experimental agent testability prediction model.

However, three major domain-related issues were hindering the idea implementation:

- **Incomplete agent measurement models**: The first issue pertains to the absence of a comprehensive agent measurements model or, at the very least, a well-defined and comprehensive framework for defining such a model. This is mostly because there is no consensus on agent definition nor agent properties list, leading to a multitude of heterogeneous, incomplete, and difficult-to-unify models.

- **Absence of a comprehensive JADE testing framework**: despite JADE being a widely used platform for over 20 years, there is a notable absence of a complete and effective testing framework specifically for JADE agents. Existing testing solutions primarily focus on interaction testing and debugging of ACL (Agent Communication Language) messages, neglecting unit-level testing.

- **Lack of benchmarks in MAS testing**: the third issue, closely related to the previous one, is the lack of benchmarks in the field of MAS testing. There are only a few open-source MASs available, and among those, almost all lack test code,

These three major challenges, lead to extending the research scoop by incorporating two additional research goals. Each goal is treated as a separate study and its outcomes are exploited to answer the main research study. Overall, three goals are established for this scientific research:

- **Goal 1**: **Develop a universal framework for agent measurement**: a framework for agent measurement model definition that transcends specific agent definitions and exploits previous research.
- **Goal 2**: **Develop a JADE Testing Framework**: a comprehensive framework for testing JADE agents at both the unit and agent levels.
- **Goal 3**: **Conduct the empirical analysis**: to investigate the agent testability

2 | Thesis outline

While the previous three research goals collectively contribute to the overarching theme, they can also be approached as independent studies. Each with its own specific research question and methodology. Consequently, in this thesis, each goal is presented in a separate chapter. This structure ensures a coherent progression of ideas, allowing for a logical and cohesive introduction of key concepts and findings. The thesis is organized as follows:

Chapter I Background and Research Methodology: The chapter is an introduction to the research domain concepts of agent, multi-agent systems, testing, and testability. The chapter also examines existing research on agent-based systems testability. Since the next chapters are presented as independent studies, the overall research methodology along with the selected sample of JADE systems are presented in this chapter.

Chapter II Unified Framework for Agent Properties Measurement (UFAPM): This chapter presents UFAPM which utilises the Goal-Question-Metric (GQM) approach (Basili et al., 1994) to specify and tailor a context-related agent properties measurement model. The chapter details both the framework's methodology for defining a measurement model and its application to the generation of a JADE agent measurement model. It also includes a validation of the defined JADE agent model on the selected sample of Multi-Agent System (MAS).

Chapter III JADE Testing Framework (JTF): The chapter presents JTF, a comprehensive solution for testing JADE agents. The framework includes unit-level testing through the UJade component, which is built upon JUnit and PowerMockito frameworks, and agent-level interaction testing via the JAT4 component. The chapter provides an empirical evaluation of JTF's usability and effectiveness. The test cases for the agents that will be examined in the next chapter are developed as a part of the study of this chapter.

Chapter IV Testability Investigation: The chapter represents the core of the thesis, focusing on the empirical analysis undertaken to address the agent testability research question.

Leveraging the solutions from Chapters II and III, the study involves static analysis of the agents under investigation source code and test code along the application of statistical methods and machine learning techniques on the collected data to answer the research questions.

Conclusion and Future work: This section summarizes the key findings and contributions of this research and sets the headlines for future works.

3 | Research Contributions

Our contributions throughout this research are summarised as follows

- **A Unified Framework for Agent Properties Measurement (UFAPM):** A framework for defining and measuring agent properties. Based on the Goal-Question-Metric (GQM) approach, the framework for tailoring a context-specific agent measurement model.
- **A JADE Agent Measurement Model:** a complete, effective, and robust model to measure JADE agent properties and attributes from source code.
- **JADE Testing Framework (JTF):** A comprehensive testing solution for JADE agents, testing at both the unit and interaction levels.
- **JADE Agent Testing Effort Measurement Model:** a model to measure agent test writing effort from the test code.
- **JADE Measurement Project Tool (JMP):** A code analyser software tool for JADE agents, provides a practical and automated way to measure more than 25 agent source code metrics and 8 testing effort metrics.
- **Benchmark of Agent Tests:** Over 431 high-quality test cases for 23 agents, available as open-source code, which can serve as a benchmark for other studies

Chapter I: Background and Research Methodology

1 | Agent & Multi-Agent Systems

The Agent-Oriented Paradigm (AOP), is presented as an advancement in software engineering, addressing the challenges of modern, distributed, and dynamic systems. This paradigm shifts the focus from procedural or object-oriented programming to designing systems in which a collection of agents interacts, collaborate, and negotiate to achieve their goals. Within the AOP, an agent is defined as an intelligent computer entity capable of perceiving its environment, processing the information, and taking actions to achieve predefined objectives (Wooldridge & Jennings, 1995). Agents are defined by their capacity to operate autonomously, making decisions based on established rules or learning mechanisms. They exhibit properties like reactivity (responding to environmental changes), proactivity (taking initiative), social ability (communicating and collaborating with other agents), and mobility (moving across different environments) (Wooldridge, 2001). These properties enable agents to operate effectively in complex, dynamic uncertain environments, providing a solid framework for developing intelligent systems.

Multi-agent systems (MASs) (a.k.a. agent based systems) are practical implementations of the AOP. They consist of multiple interacting agents that work together to solve problems that are beyond the capabilities of a single agent. In a MAS, agents can communicate, cooperate, negotiate, and coordinate with each other to achieve individual or collective goals. The collaborative nature of MAS and its distributed architecture make it an ideal choice for handling complex, large-scale problems like chain management, smart grids (Vahi & Ignise, 2024; Wang, 2024; Shobole & Wadi, 2021), automated trading systems (Hernes et al., 2024; S. Chichin et al., 2014), and autonomous vehicles (Petrillo et al., 2018)…etc.

2 | JADE Platform for Agent Development

The Java Agent DEvelopment Framework (JADE) is an open-source platform intended to facilitate the development of multi-agent systems (Bellifemine et al., 2007). Implemented in JAVA, JADE provides a middleware that complies with the FIPA (Foundation for Intelligent Physical Agents) specifications[1], ensuring interoperability among MASs. JADE provides a range of features that facilitate the development, deployment and management of MASs like peer-to-peer agent communication via asynchronous message passing, yellow pages' service that supports publish-subscribe discovery mechanisms, and a graphical tool for debugging and deploying agents. A key advantage of JADE is its flexibility and scalability. A JADE-based system can be distributed on multiple machines and agents can move between them. JADE is widely used for agent-based systems

[1] http://www.fipa.org

development, in both academia and industry (Wrona et al., 2023; Bergenti et al., 2020; Kravari & Bassiliades, 2015).

3 | Agent-based systems Testing

Software testing refers to the set of activities (planning, preparation, execution, reporting) conducted to discovery and evaluation of properties of the software (ISO, 2022). Its purpose is to verify that the system behaves correctly, free from errors or defects, and following its development requirements. It involves executing the software under a controlled condition (test case) under which a tester will determine whether it is behaving as expected. A test case specification includes detailed steps to execute (manually or automatically), the input data, and the expected results (ISO, 2022). Test cases can be designed to cover various aspects of the software's functionality, performance, and security.

Commonly MAS testing is approached on five distinct levels, as outlined by Nguyen et al. (2009).:

- **Unit-level testing**: This initial phase involves the examination of individual units that constitute an agent, such as its goals, knowledge base, plans, and reasoning engine. Each component is tested to ensure it functions as intended, laying a solid foundation for subsequent testing phases. This level is crucial for identifying and rectifying defects at the earliest stage of the development process.

- **Agent-level testing**: This phase agent as one entity is tested, it is seen as an integration test for the previously tested units. Here, the agent's ability to achieve its goals and interact with its environment is verified. This level of testing ensures that all units work together seamlessly within the agent, providing a clearer picture of the agent's overall functionality and performance.

- **Integration or Group-level testing**: At this stage, the focus shifts to the interactions among a group of agents. This level is essential for assessing the emergent properties and collective behaviour resulting from these interactions. By testing agents' collaboration, developers can identify potential issues in group dynamics and ensure that the agents can cooperate effectively to achieve common goals.

- **System or Society-level testing**: This phase examines the agent society, focusing on the emergent and macroscopic behaviours of the entire system. This level assesses the quality properties, such as openness, fault tolerance, and Auto-X properties (self-organization, self-healing, and self-optimization), are tested at this level.

- **Acceptance testing:** The final phase involves evaluating the acceptability of the MAS. This level ensures that the system meets the required standards and user expectations, confirming that it is ready for deployment. Acceptance testing validates that the MAS can perform its intended functions in real-world conditions, providing confidence that the system will deliver reliable and effective performance.

Throughout this thesis, the testability is investigated at the first two levels, unit and agent levels, only.

4 | Agent-based Systems Testability

What is the software testability? Many definitions were proposed in the literature. For Binder (1994) it is about the controllability and observability of the software "*To test a component, you must be able to control its input (and internal state) and observe its output*". For Voas (1992) it is the software's sensitivity to faults, "*the probability that a piece of software will fail on its next execution during testing, provided it contains a fault*". ISO (2011) defines it as the "*degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met*". NASA reliability and maintainability standard (NASA, 2017) defines it as "*design characteristic that permits timely and cost-effective determination of the status (operable, inoperable or degraded) of a system or subsystem with a high level of confidence. Testability attempts to quantify those attributes of system design that facilitate detection and isolation of faults that affect system performance*".

These different perceptions of testability are the result of its multifaceted nature. Binder presented a part of these facets (1994) using a fish-bound diagram(Figure I-1). The latter identifies six major facets: the representation (documentation), the implementation, the built-in test, the test suite, the test support environment, and the software process; each is associated with factors that ease or curb testing.

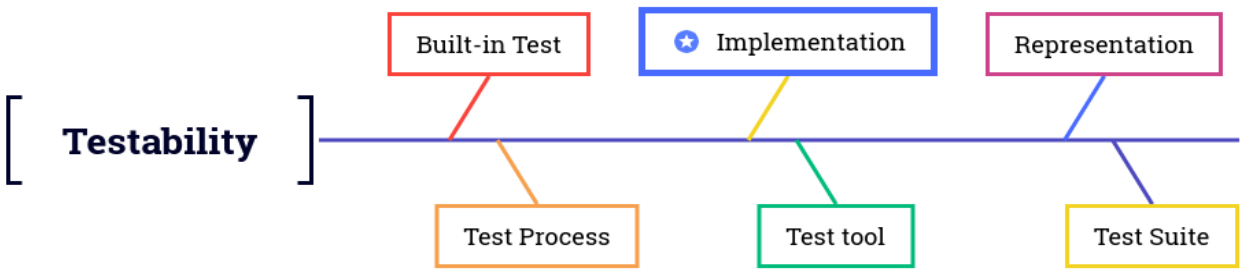


Figure I-1 Binder's Testability facets

In this thesis the focus is on the implementation facet, therefore, throughout this thesis testability is used to mean ease of testing: it is "*the attributes of software that bear on the effort needed to validate the software product*". (ISO, 2001)

The testability of software in general and object-oriented (OO) software in particular, are thoroughly studied in the literature. A large number of techniques and metrics have been proposed by both practitioners and researchers to understand, measure, predict and improve it thought the software development process. For instance, software testability was related to quality of requirements and specifications and how easy to generate test from theme (Zakeri-Nasrabadi & Parsa, 2024; Hayes et al., 2015; Izosimov et al., 2012), metrics for requirement readability along machine learning technique were used to predict the overall software testability (Hayes et al., 2015)

Alternatively, works like (Efatmaneshnik et al., 2018; Efatmaneshnik & Ryan, 2017; Baudry & Le Traon, 2005; Baudry et al., 2004) looked into the relationships between software design and architecture, and software product testability. The results include the identification testability anti-patterns (Baudry & Le Traon, 2005) setting the best practices and architectures for designing for testability (Ramirez et al., 2024; Meixner & Gullo, 2021; Payne et al., 1997) and suggesting testability measurement models based on design metrics (Zakeri-Nasrabadi et al., 2024; Abdullah et al., 2015; Khan & Mustafa, 2009).

Furthermore, software code and tests were also heavily investigated to study testability. Either to measure quantitatively and qualitatively testability via the use of test oriented metrics Number of Test cases, Number of Test Lines of Code (TLOC), Number Of Assertions …etc (Toure et al., 2014; M. Badri & Toure, 2012; Bruntink & Deursen, 2006). Or understand the relationship between software product internal properties and testability; properties like Size, Complexity, Inheritance coupling (Bajeh et al., 2020; Ouellet & Badri, 2019; Touré et al., 2018), cohesion (L. Badri et al., 2010), control-flow dependencies (M. Badri & Toure, 2012) on testability …etc.; and eventually to proposing testability prediction models using machine learning (Ouellet & Badri, 2024; Albattah, 2022; Moudache & Badri, 2022; Matcha et al., 2022; Boucher & Badri, 2018; Touré et al., 2018; Vig & Kaur, 2018; M. Badri et al., 2017; L. Badri et al., 2011). A summary of the entire state-of-the-art and state-of-the-practice on the testability of object-oriented (OO) software , until 2019, can be found in (Garousi et al., 2019).

However, unlike traditional OO software, agent-based software testability is little addressed, only three research papers were identified (Gonçalves et al., 2019; Winikoff, 2017; Winikoff & Cranefield, 2014), which is surprising given the importance of agent-based software verification and the persistent interest in testing as a way to achieve that (Baudet et al., 2019; Nascimento et al., 2017; Cunha et al., 2015; Bagić Babac & Jevtić, 2014; Carrera et al., 2014; Nguyen et al., 2010; Tiryaki et al., 2007; Coelho et al., 2006; D. A. Ostrowski & R. G. Reynolds, 1999; Low et al., 1999; Van Liedekerke & Avouris, 1995). Furthermore, testing is one of the most used techniques for agent system software verification (Bakar & Selamat, 2018).

(Winikoff & Cranefield, 2014, 2015) are the first works discussing the testability of multi-agent systems. The papers' authors wanted to investigate whether is it feasible to assure belief-desire–intention (BDI) agent systems effectiveness by testing, and how factors like the size and shape of the program and the existence of failure handling function may affect the system testability. They defined testability as the testing effort, caught by the number of test cases, required to test a BDI-agent program to the satisfaction of all paths test adequacy criterion.

By considering BDI-agent execution as a data transformation from a goal-plan tree to a sequence of action executions, Winikoff et Cranefield counted the number of behaviours, both successful and unsuccessful (i.e. failed), for a given goal-plan tree. They found that the space of possible behaviours for BDI agents is indeed large and therefore testing systems as a whole by considering all paths is likely to be infeasible. Furthermore, the size of the behaviour space, and subsequently the number of test cases, strongly depend on the depth and width of the goal-plan trees and the presence of failure handling functionality. So, they suggest keeping BDI goal-plan trees shallow and sparse and/or avoiding failure handling to ensure testing feasibility.

Later Winikoff (2017) revisited the previous questions by considering a less stringent test adequacy criterion: all edges test adequacy criterion. In this second study, it was found that the number of tests required to satisfy the edges coverage was very low, and unlike his previous work, disabling failure handling did not make a big difference to the number of required tests, as result it was concluded that test BDI-agent systems are feasible when using all-edges test adequacy criterion.

Inspired by the previous work, Gonçalves et al. (Gonçalves et al., 2019, 2022; Machado & Gonçalves, 2020) proposed a method to assess the testability of the social dimension of MAS specified with the organisational model MOISE (Model of Organisation for multI-agent SystEms) (Hannoun et al., 2000). In the adopted workflow a MOISE specification of the system organisation is mapped to a Coloured Petri Net (CPN)(Jensen, 1997). The number of paths in both the CPN and its associated state graph (reachability graph) are considered as the number of test cases required to validate this specification under the coverage criteria: all paths and state transition paths respectively.

These works on agent-based system testability are limited by their investigation scope, only factors that influence the number of test case generation (size) were considered. The latter metric was used as an indicator of the testing effort, yet few does not always mean less effort, a test case could be complex that its implementation became infeasible (in the available time and budget). In addition to that, only the system as a whole testability was studied, but what about agent testability? Agent testing, at both unit and interaction levels, as mentioned before, is required for any successful testing strategy for agent-based systems (Nguyen et al., 2009). Besides, the studies' conclusions and generalisations are bound to their initial assumptions on both agent and system architectures: BDI-

agent in all the studies and MOISE-based society in Gonçalves et al.'s work (Gonçalves et al., 2022, 2019; Machado & Gonçalves, 2020).

5 | Research Methodology

The aim of this D.Sc. research is to investigate the testability of the main building block of an agent based system, that is agent; to understand how an agent properties and attributes can affect its testing effort, and to examine to which extend that effort can be predicted from the agent source code. The testability is studied from the perspective of agent test writing effort at the first testing levels unit and agent. It has as a subject JADE multi agent systems. The choice of this type of system rather than other platforms based systems, is driven by two facts: First, JADE is agent-architecture independent platform which leans to make the obtained result more generalised; and second JADE platform is one of the most used platforms for agent-based software systems development in both academia and industry, and more essentially

The initial research methodology was designed to be process oriented (Figure I-2). It consisted of 3 stages. The first, on a sample of multi agent systems, agents' properties and attributes is characterized and measured. Then in the second stage agent testing effort is characterised and measured form the selected agents' tests code. And lastly the collected data are statistically analysed using techniques like correlation analysis, logistic regressions and machine learning to answer the research questions.

Figure I-2: Initial Research Process

However, three major issues prevent the execution this process:

- **The lack of a complete and reliable agent measurement model**: After conducting a specialized literature review, it was found that many proposed works generally used a hierarchical decomposition approach, refining agent properties into attributes (capabilities) and metrics. This approach is known to have ambiguity issues within the decomposition process (Deissenboeck et al., 2009). Moreover, only a subset of agent properties is targeted each time, and the works that target the same properties do not always share neither the same definition of these properties nor the same agent definition

(reactive vs. cognitive). This resulted in highly heterogeneous models and a lot of metrics propositions. To the best of our knowledge, no complete comprehensive measurement model for agent properties exists. Furthermore, the lack of standard definitions of agent and agent properties, such models may not exist even. Alternatively, a more general and fixable approach will be suitable.

- **The fail to build a reliable sample of MAS**: The search of code repositories, like GitHub, GitLab, SourceForge, and Google Code Archive, for open source MAS projects, resulted in the identification of only a few open-source MASs, almost all of them lack test code.

- **The absence of an effective JADE agent testing solutio**n: Again the literature and the previous code repository review revealed that existing solutions primarily target agent interaction debugging and testing, and unit testing is largely overlooked.

Due to these issues, the research scope had to be extended and the research methodology had to be reviewed to accommodate the new scope. The new revised methodology is a goal oriented and aiming to achieve the following 3 gaols:

- **Goal 1**: **Develop a universal framework for agent measurement**: a framework for agent measurement model definition that permit to tailor a context-specific agent properties measurement model.

- **Goal 2**: **Develop a JADE Testing Framework**: a comprehensive framework for testing JADE agents at both the unit and agent levels.

- **Goal 3**: **Conduct the empirical analysis**: to address the primary research questions:

    - what agent's property influences its testability, and on which testing level (unit or agent) does it do?

    - what agent's attribute (capability) influences its testability and on which testing level (unit or agent) does it do?

    - could agent testing effort be predicted from its source code?

The focus of in the new methodology was on minimising as much as possible the threats of validity of the overall research. The rationale behind the first goal where a generic framework for agent measurement definition needs to be developed, instead of just proposing directly a new measurement model for JADE agent, is to mitigate as much as possible the validity threats that can be associated with the construction of such of model. The framework will give the objective

background that will support the JADE agent measurement model definition, by permitting to identify the full relationship between agent properties and metrics, to determine what attributes and metrics were missing or inconsistent, and to provide a context for interpreting the measurements later.

The second goal, it is achievement will allow not just to building the required test cases for agents within the selected sample of JADE MAS, but subsequently to improve the quality of the main study (the 3rd goal) and the trustworthiness of its results, since all agents' test code will be developed using the same framework.

Lastly, in the third goal the outcome of the previous ones will be utilised to execute the original plan and to answer the main research questions.

Figure I-3, illustrates the overall adopted research process. The methodology employed to achieve each of the three goals will be thoroughly detailed in their respective chapters.

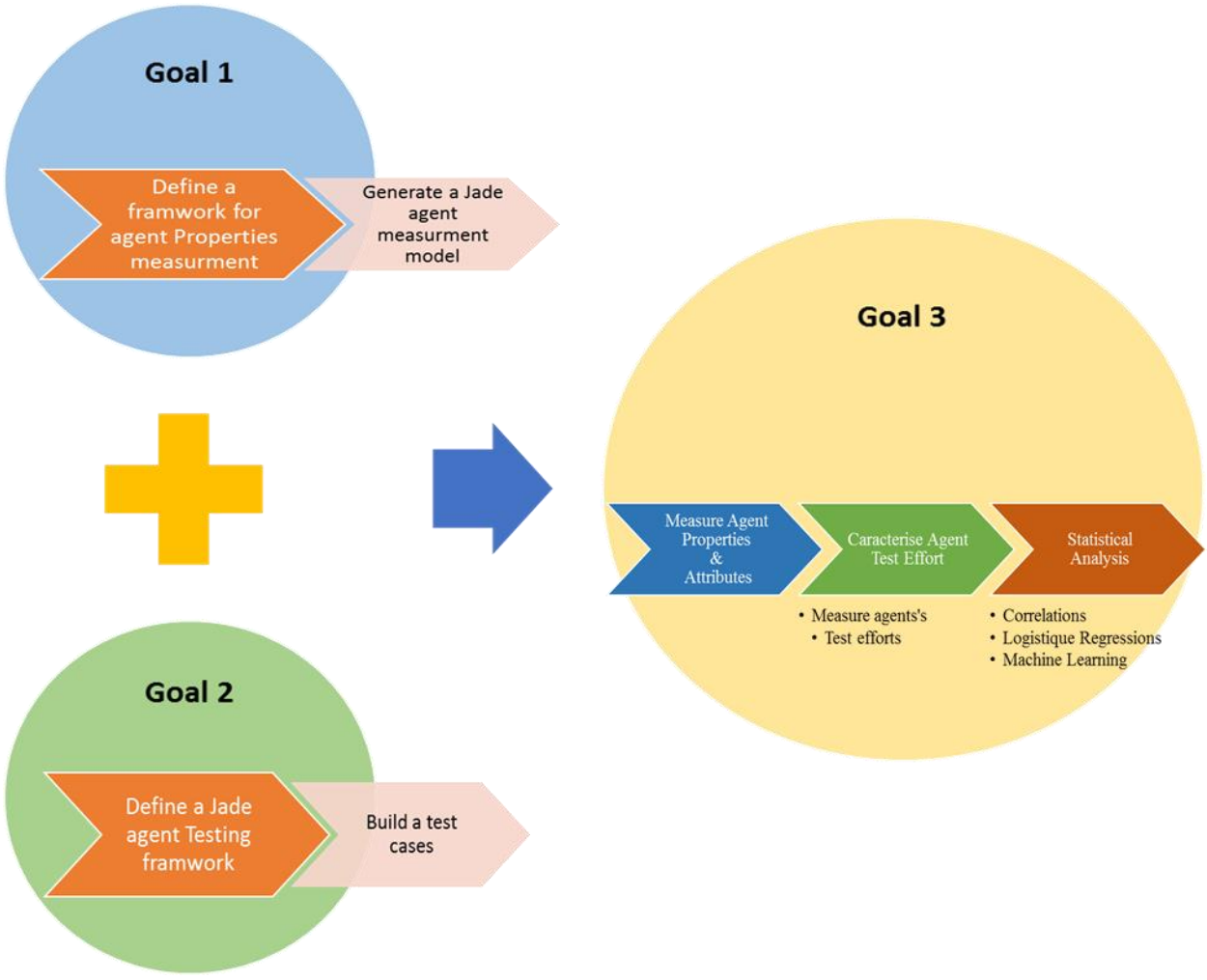


Figure I-3: The overall adopted research process

6 | Systems Under Study

Through this thesis, and it's inner three studies, seven JADE multi-agent systems containing 23 agents, are used as a sample. Four of them are from the GitHub repositories (MAS 1,3,6,7). The

MAS 2 was developed as a case study in the JAT paper (Coelho et al., 2007). The MAS 4 is from the JADE platform distribution package and the MAS 5 is from the google code project archive. Some basic statistics about these systems are presented in Table I-1. For each system, the number of agents in the system, the sum of agents' lines of code and the agent code cyclomatic complexity (CC, calculated by EclEmma[2]) are given. Systems 1, 2 and 7 are relatively small, meanwhile, systems 3 and 5 have medium sizes. The mean number of agents in the previous systems is 2. However, system 7 is the largest and the most complex one, with 9 agents and a total agents' cyclomatic complexity is 206.

MAS	System	Agents	LOC	CC
1	Auction System -1-	2	322	29
2	Auction System -2-	3	397	47
3	Auction System -3-	3	525	79
4	Book Trading System	2	266	52
5	Hospital-Appointment Allocation System	2	520	100
6	Home Automation System	9	1972	206
7	Treasure Hunt System	2	254	46
	Total	**23**	**4256**	**559**

Table I-1 The MASs understudy

The first system[3] (Figure I-4) is an auction system (reverse auctions) implemented using agent technology. The system consists of two agent types, a company and a carrier agent. The company agent is looking for an agent to do a job with the lowest possible price. Meanwhile, the carrier agent is a bidder agent, who tries to get the highest possible value for doing a job. When there are multiple bidders, they will underbid each other, in the hope of others to forfeit. Both the company and the carrier agents are implementing FIPA iterated contract net protocol[4].

The second system is another implementation of the previous auction system (Figure I-5); this time the system contains three agent types. The enterprise agents sell some goods; the bargainer agent wants to buy an item with the lowest possible price. The difference with the first system is that once the bargainer identifies the candidate enterprises, it enters into negotiation with them, in the hope to get a better deal via an intermediate agent, the auction agent.

[2] https://www.eclemma.org/ni
[3] https://github.com/Adrrei/Multi-Agent-Systems-In-JADE
[4] http://www.fipa.org/specs/fipa00030/PC00030D.html

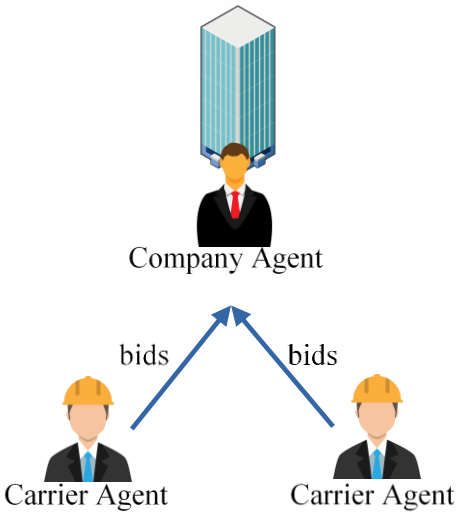


Figure I-4: Auction System -1

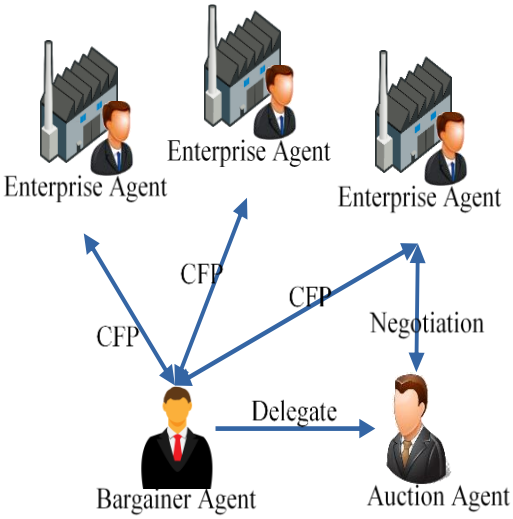


Figure I-5 Auction System -2-

The third system[5] implements a first-price sealed-bid auction Figure I-6) in which bidders place their bids in a sealed envelope and simultaneously hand them to the auctioneer, who announces the bidder with the highest bid as a winner. The system has three agent types: Auctioneer, BidderComp, and BidderHuman. The auctioneer agent is responsible for the main auction events. BidderComp and BidderHuman are both the bidder agents, whereas BidderComp is an autonomous agent. It uses the ALL-IN as a bid strategy that is always bid on any item introduced in an auction, using the available budget (money). Once its money runs out, it terminates. Meanwhile, the BidderHuman agent is controlled by a human user via GUI. The user can add more money to the agent's wallet, send bids or refuse to participate in an item auction.

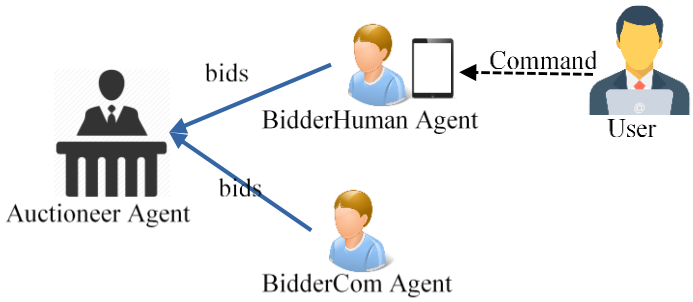


Figure I-6 Auction System3 (First-Price Sealed-Bid)

The fourth system is the traditional book trading system available with the JADE platform distribution[6] (Figure I-7). The system comprises two types of agents: BookBuyer and BookSeller agents, where the BookBuyer agent tries to buy a book at a low price, from one of the available BookSeller agents.

[5] https://github.com/ardiyu07/jade-blind-auction
[6] https://jade.tilab.com/download/jade/

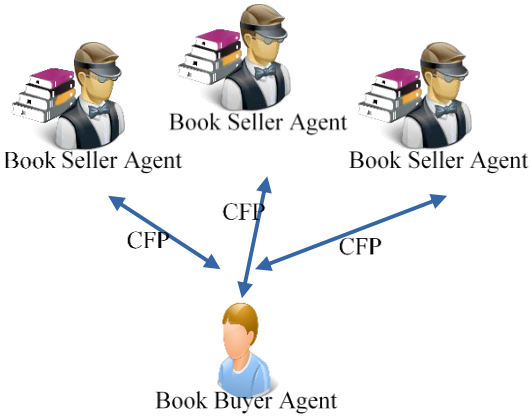


Figure I-7 Book Trading System

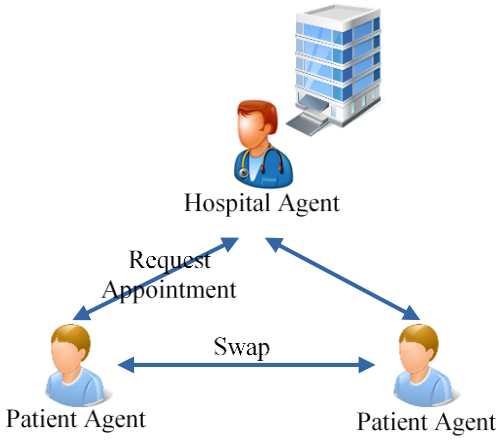


Figure I-8 Hospital-Appointment Allocation System

The fifth system[7] is a simple hospital-appointment allocation system (Figure I-8). It contains two types of agents: a hospital agent responsible for appointment allocation and a patient agent who tries to get an appointment that matches its preferences. The patient agent can also ask other patients for appointment swaps (to get a preferred one).

The sixth system[8] was developed as part of a home automation solution project; with the JADE framework as an intermediate layer to manage interactions with physical hardware devices, such as sensors and actuators (Figure I-9). Each physical device (light bulb, button, infrared remote-control receivers, temperature or brightness sensors, etc.) is associated with an agent that acts as a wrapper of this device, abstracting the way of interacting with these devices. The hardware devices are connected physically to a set of interconnected microcontrollers, altogether, they form a mesh topology network.

The system environment consists of buildings, each building contains some rooms, and each room may contain one or many devices. The buildings and the rooms are also agents.

The current system version contains nine (9) types of agents, a demo agent responsible for system launch, a controller agent to interact with a user (display the system state and receive user commands), a room agent, a building agent, a bulb agent and a toggle switch agent, two sensors agents (light and temperature sensors) and a mesh Net gateway agent to interconnect the software and the hardware. This case study is characterized by the complexity of its agents and the complexity of their interaction scenarios. Agents like the bulb agent, temperature and light sensors are showing high values in most of the properties (good values in worst cases).

[7] https://code.google.com/archive/p/mascoursework/
[8] https://github.com/AL333Z/JadeHomeAutomation

The last system[9] simulates a treasure hunt game. The system comports two agents (Figure I-10). The first one is a "game master" that hides a treasure on a grid. The second one is a "player" that tries to find the treasure. At each run cycle, the player agent moves by one square on the grid, and the game master tells it whether it is getting closer or further away from the treasure.

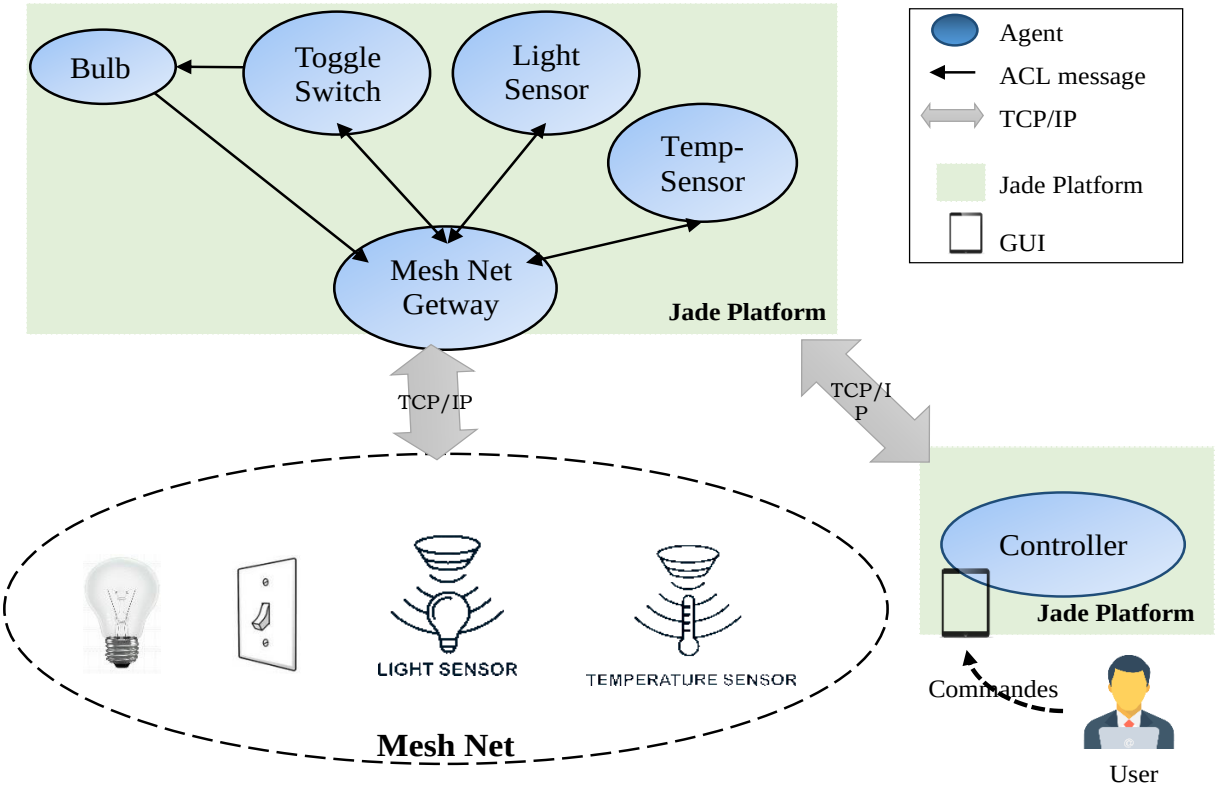


Figure I-9 Home Automation Architecture[10]

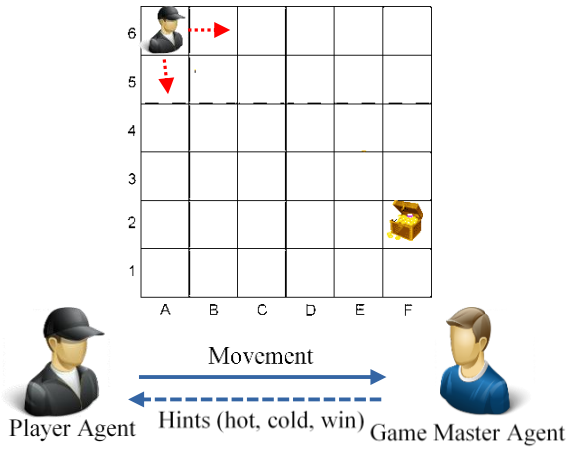


Figure I-10 Treasure Hunt System

.

[9] https://github.com/Syncrossus/JadeTreasureHunt
[10] https://github.com/AL333Z/JadeHomeAutomation

Chapter II: A Unified Framework for Agent Properties Measurement

1 | Introduction

In this chapter, the primary interest is on agent measurements. It looks into understanding how agent properties (reactivity, pro-activity, sociability, autonomy, mobility, rationality, learning and adaptability (S. Franklin & Graesser, 1997; Wooldridge & Jennings, 1995)) can be characterized and measured, in aim to define a comprehensive JADE agent measurement model.

Many works have been done in the area of agent-based systems measurement in general and agent measurement in particle, ranging from proposing quality models (where software product's external quality attributes are related to agent's internal quality attributes), proposing agent properties measurement models (Sivakumar & Vivekanandan, 2012; Alonso, Fuertes, Martínez, et al., 2010), adapting object-oriented (OO) and procedural metrics to the agent-oriented (AO) paradigm (Far & Wanyama, 2003); and proposing new metrics adapted to the AO paradigm (Hrabia et al., 2015; Lass et al., 2009; Klügl, 2008; Dumke et al., 2000). A good review of methods and techniques of agent-based systems measurement and evaluation is presented in (Dumke et al., 2009).

The review of these works in aim to identify or at least build a complete JADE agent measurement model revealed that the lack of standardisation of agent notion resulted in different perceptions of agent, its properties and how it can be measured. The identified agent measurement models are heterogeneous and incomplete. Each time only a subset of agent properties is targeted and even the models that include the same properties do not always share neither the same definition of these properties nor the same agent definition. Furthermore, generally a hierarchical decomposition approach is used to defined these models, refining agent properties into attributes (or capabilities) and metrics. But this approach is known to have ambiguity issues within the decomposition process (Deissenboeck et al., 2009) and cannot prove that the adopted metrics are sufficient and complete. To the best of our knowledge, no complete comprehensive measurement model for agent properties exists.

Due to the lack of standard definitions of agent and agent properties, we believe that conceiving one solution that fit all is not feasible, thus throughout this chapter a more general and fixable approach is adopted. The unified framework for agent properties measurement definition (UFAPM), is built and proposed to define JADE's agent measurement model. Built upon the results of previous agent measurement works, UFAPM uses the Goal-Question-Metric (GQM) approach (Basili et al., 1994) to specify and tailor a context-related agent properties measurement model.

The chapter is outlined as follows. Section 2 | presents the agent properties measurement's state of the art; followed by a description of the unified framework for agent properties measurement (UFAPM) in section 3. Sections 4, focuses on the implementation of UFAPM for the definition of a JADE's agent measurement model. Section 5 presents a source code analysis tool JMP (JADE

Measurement Project) supporting the defined JADE model. The tool is used next in section 6 on the set of JADE-based MAS, introduced in the previous chapter (Chapter I: 6 |), to evaluate the accuracy of the adopted JADE's agent measurement model. Both the developed framework and the defined measurement model threats of validity are assessed in section 7.

2 | State of Art

Looking for research on agent-based systems and agent measurement, the specialized literature is dominated by works directed at agent runtime quality measurements like performance, utilities, fitness and communication (Papoudakis et al., 2021; Bernstein et al., 2019; Sanislav et al., 2018; Quintero-Parra et al., 2017; Zuparic et al., 2017; Rouhani & Mirhosseini, 2015; Hamada & Sugawara, 2013). This is interpreted by the fact that developers were trying to understand how well their agent-based solution was performing in runtime (Dumke et al., 2009). However, in the last decade, along with more adoption of agent-oriented (AO) paradigm in academia and industry (Leitão & Karnouskos, 2015; Müller & Fischer, 2014) and the considerable progress within agent-oriented software engineering (AOSE) practices (Cossentino et al., 2014; Sturm & Shehory, 2014a, 2014b), the focus shifted into agent-based software quality definition and measurement, particularly agent properties assessment. In the following, we provide insights into some of these works.

The first early works were (Wille et al., 2004; Dumke et al., 2000), where the authors argued that agent-based software quality evaluation and measurement should consider product, process and resource aspects of the software at both levels of abstraction: agent level and system level. For instance, at the agent (as product) level, they suggested measuring: agent documentation quality (completeness, readability, etc.), agent design (size, complexity and functionality appropriateness), and agent behaviour-related properties (communication, interaction, collaboration, cooperation, negotiations learning, adaptations, and specialisation). In these works, Dumke et al. suggested a list of informal metrics that can be applied at each measurement level but without providing details on the way to compute them. However, they claimed that, in most of the cases, object-oriented (OO) metrics can be reused (Dumke et al., 2000), since the AO paradigm was considered as the evolution of the OO paradigm (Wooldridge, 2001).

In the years between 2008 and 2011, Alonso and his co-workers published a series of works (Alonso et al., 2011; Alonso, Fuertes, Martínez, et al., 2010; Alonso, Fuertes, Martinez, et al., 2010; Alonso et al., 2009, 2008) on agent sociability, pro-activities and autonomy measurement. Their objective was to set up a quality model considering agent specificity. They proposed a hierarchical quality model, where they have refined agent properties into attributes measured using OO and AO metrics. For example, sociability was refined into communication, negotiation and collaboration attributes; autonomy into self-control, function independence and capability attributes; and pro-

activity into initiative, interaction and reaction attributes. The proposed model is supported by a software measurement tool for the JADE agent (Soza, 2018).

Following the philosophy of Alonso, two additional models were proposed to measure reactivity (Sivakumar & Vivekanandan, 2012), and intelligence properties (Mahar & Bhatia, 2014). The former was associated with interaction, perception and communication attributes. Whereas the latter was associated with adaptability, learnability, and pro-activity.

Alternatively, the Goal-Question-Metric (GQM) approach was used in (Di Bitonto et al., 2010, 2012) to suggest a multi-agent system (MAS) measurement plan independent of implementation details. The proposed plan has five measurement goals; four of them are the agent properties: autonomy, reactivity, rationality and adaptability and, the fifth is the environment complexity. For each goal, a set of questions and metrics were identified to assess the MAS at both the agent level and system level. Most of the proposed agent-level metrics are subjective rating metrics.

Sivakumar et al. (2011) developed an automated software tool to measure JADE's agents: interaction, communication, collaboration, learning, adaptability, mobility and specialisation. The proposed measurement model had a flat structure, i.e. no distinction between properties and attributes, all of them were measured directly using metrics, in most of the cases dynamically. However, no details on the model definition process were given.

In 2016, a quality model for MAS named (QM4MAS) was proposed (Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016). It is an extension of the ISO-9126 (ISO, 2001) software quality model to support agent-based software specificity. The authors of this model attempted to give a global view of MAS quality by relating the external software characteristics (functionality, reliability, usability, etc.) to the agent characteristics (seen as sub-characteristics in this model). QM4MAS defines eight (8) software characteristics, with eleven (11) sub-characteristics of which eight are the agent properties: autonomy, reactivity, pro-activity, sociability, rationality, adaptability, granularity and specialisation. In contrast to previous works, the agent properties (sub-characteristics) in QM4MAS were assessed directly (without being refined into attributes) by a set of metrics, mainly dynamic (Marir, Mokhati, Bouchelaghem-Seridi, et al., 2016).

In 2020, Benaboud and Marir (Benaboud & Marir, 2020) proposed an upper ontology for MAS properties measurement, where the system properties can be decomposed into a set of measurable characteristics, associated with system components (agents, environments, organization, etc.). For each component, the implementation details (the procedures and techniques used to implement the related characteristics) are used to define the characteristics' measurement metrics. The authors used the previous upper ontology to instantiate a flexibility ontology, where for agent

component, the flexibility was associated with the agent's: reactivity, proactivity, social-ability, learnability, adaptability, and fault-tolerance. The previous agent properties (named characteristics) were assessed directly by a set of dynamic and static metrics.

Apart from studying agent property measurements from a quality measurement perspective, there were several works for context-related measurements of agent properties. These works can be classified into two categories: properties that are measured directly with predefined metrics, like in the case of interaction (Joumaa et al., 2008), communication (Gutiérrez & García-Magariño, 2009), reactivity (Dam et al., 2013); and properties that are seen as multifaceted notions, measured indirectly based on values of their associated attributes, for instance: autonomy (Antsaklis, 2020; Hrabia et al., 2015; Mostafa et al., 2014; Huber, 2007; Hexmoor et al., 2003; Huang & Nof, 2000; Barber & Martin, 1999; Brockhoff & Schmaul, 1996), complexity (Marir et al., 2014; Sarkar & Debnath, 2012; M. Mala & İ. Çil, 2011; Klügl, 2008; Dziubiński & Verbrugge, 2007; Dekhtyar et al., 2002; Far & Wanyama, 2003), and intelligence (Hasebrook et al., 2002; D. Franklin & Abrao, 2000). Table II-1, presents the results of our specialized literature review. Metrics names are detailed in Appendix 1.

Property	Papers	Attributes	Metrics
Adaptability	(Sivakumar & Vivekanandan, 2012; Sivakumar et al., 2011)		KUP, KUG, WMC, EHF
	(Di Bitonto et al., 2012)	Respond to external stimuli Manage different situations	CorrChanReac*, RightRol* LearAb*, EurFinAb*, ExcMaAb
	(Mahar & Bhatia, 2014)		WMC, EHF
	(Alonso et al., 2009)	Evolution capability	SUC, FSU
	(Benaboud & Marir, 2020)		FBC, FSC, FEC
Granularity	(Marir et al., 2014)	Structural complexity Behaviour complexity	SAS, ASG, DAS BSA, ACB, ANSB
	(Sarkar & Debnath, 2012)		Nei, Nci, Npi, Nki, Nsi Nresi, NIIIi, NIICi, NIIIi, NOIi
	(Klügl, 2008)		ACR*, APM, SPK, NCR
	(Alonso et al., 2009)	Self-control	SC, ISS, BC
	(Di Bitonto et al., 2012)	Complexity of interaction	CoopGard*, CompGard*, Tr&RepMod*
	(Far & Wanyama, 2003)		CCM, UF*
	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		BG, KG
Autonomy	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		RLRSs, RRA
	(Alonso, Fuertes, Martínez, et al., 2010; Soza, 2018)	Self-control Functional independence Evolution capability	SC, ISS, BC EMR SUC, FSU
	(Huber, 2007)	Social independence Social integrity	SI SD
	(Di Bitonto et al., 2012)	Proactivity Autonomy in organisation structure	NegAg*, ComAutAb* MoreRol*, DiaErPrAb* PosStr*, SharTask*
	(Arora & Sasikala, 2016)	Proactivity Autonomy within organisation	NegAg2, Communi-cation, MoreRol2, DiaErPrAb* PosStr2, SharTask2
	(Antsaklis, 2020)		AL

Property	Papers	Attributes	Metrics
	(Huang & Nof, 2000)		DAA
	(Hrabia et al., 2015)	Perception and acting skills	PAscore
		Beliefs & reasoning	BRscore
		Goals & planning	GPSscore
		Learning	LScore
	(Braynov & Hexmoor, 2003)	Preference autonomy	DPA
		Choice autonomy	DCA
		Decision autonomy	DDA
		Individual autonomy	DIA
	(Barber & Martin, 1999)		R_a
Learning	(Mahar & Bhatia, 2014; Sivakumar et al., 2011)		AHF, KUP, KUG, VD
	(Benaboud & Marir, 2020)		FCG, DNG, RN_gU
Pro-Activity	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		RAP, TAP
	(Alonso et al., 2011; Soza, 2018)	Initiative	NOR, NOG, MAG
		Interaction	MAG, MC2
		Reaction	NPR, AOC
	(Di Bitonto et al., 2012)		ComAutAb*, NegAg*, DiaErPrAb*, MoreRol*
	(Arora & Sasikala, 2016)		MoreRol2, NegAg2 Communication, DiaErPrAb*
	(Benaboud & Marir, 2020)		RPA, RRG
Reactivity	(Sivakumar & Vivekanandan, 2012)	Interaction	MC, NMT
		Communication	RFM, IM, OM
		Perception	KUP, KUG
	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		RSC, TRC
	(Di Bitonto et al., 2012)	Effectiveness in perception acquisition	AgEffAcqPerc*
		Rapidity in response in a timely fashion	PercQual*, DefBeh*, insMod*, ComMin*
	(Benaboud & Marir, 2020)		CP, FP, RT
Sociability	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		RII, RIU, RU
	(Alonso et al., 2008; Alonso, Fuertes, Martínez, et al., 2010; Soza, 2018)	Communication	RFM, AMS, IM, OM
		Cooperation	ASAd, SRR
		Negotiation	AGA, MbReq, MtReq
	(Benaboud & Marir, 2020)		NIP, NRM, FRC, FRsC, EE
Rationality	(Di Bitonto et al., 2012)	Action choice mode	AgType*, PlaConst*, LearnAb*; InsMod*
		Maximisation of success	ActMax*
	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		GAA, GAUR
Specialisation	(Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)		ARA
	(Sivakumar et al., 2011)		WMC, NR
Mobility	(Sivakumar et al., 2011)		AES, SMS, AMS

* Subjective metrics

Table II-1 Agent Properties Measurements - Related Works

The literature review has revealed some problems and limits within existing works. Firstly, the absence of consensus on agent definition and agent properties list influenced the way an agent is measured. Most proposed works do not share the same definition of agent, not even the same agent properties list. These differences have led to heterogeneous measurement models difficult to unify.

Besides, most of the works have addressed only a few agent properties and, until now, there is no complete comprehensive measurement model for agent properties.

Secondly, the hierarchical decomposition approach used to characterize agent properties (properties are defined in terms of attributes and metrics) suffers from ambiguity issues: the lack of precise decomposition criteria that determine how high-level concepts (in our case properties) are decomposed (into attributes) (Deissenboeck et al., 2009); the lack of clarity on whether the metrics are sufficient and complete to assess the related attributes (or properties in case of (Marir, Mokhati, Bouchlaghem-Seridi, et al., 2016)).

Also, more than 120 metrics were identified as a part of this research, but in most of the cases measurement methods were not specified, nor the paradigm or the implementation languages under which they were proposed have been specified. Sometimes, the proposed metrics do not adhere to agent programming principles (APP) for example, the attribute hiding factor metric (AHP) proposed in (Mahar & Bhatia, 2014) to measure intelligence; and the number of public information (NUI) and the number of external accesses (NEA) metrics proposed in (Klügl, 2008) to measure agent interaction complexity. These metrics are defined respectively as the number of private variables (agent knowledge), the number of public variables and the number of public methods (action) an agent has. All these definitions are against APP. Since an agent is supposed to exhibit autonomy over its state (knowledge and actions) and to protect it from external access (Wooldridge, 2001; Wooldridge & Jennings, 1995), that is all agent variables and internal methods are supposed to be private.

Finally, due to the diversity of MAS implementation paradigms (for example: OO, knowledge-based systems, etc.) and the variety of programming languages, we believe that a one-size-fits-all measurement model is not realistic. Alternatively, a more general and fixable approach is suitable.

The next section presents a proposition of a framework for agent properties measurement model definition, where a measurement model can be specified and tailored to suit a specific measurement context (agent definition and agent implementation details, etc.). The framework development was driven by the following concerns:

- Flexibility: it should be supple enough to cope with the variety of viewpoints on agent and agent's properties notions and its diverse implementation context.
- Clarity of measurement inference process: how agent properties are identified, associated with agent attributes, and measured using well-defined and well-adapted metrics.

- Tractability between what is being measured and agent properties.
- Metrics soundness: the obtained metrics and measurement models are valid for the pre-selected agent measurement context.
- Metrics completeness: the obtained metrics are complete and sufficient to quantify the properties under measurement.
- Validity and applicability of measurement methods.

3 | Presentation of the UFAPM

The Unified Framework for Agent Properties Measurement (UFAPM) is built upon the Goal-Question-Metric (GQM) approach (Basili et al., 1994; Solingen & Berghout, 1999). GQM is a goal-oriented measurement paradigm for software engineering, where the organisation's goals are defined, refined into questions and evaluated using measurement. Starting from an operational goal, a set of questions is formulated in such a way that answering these questions will lead to the fulfilment of the goal. Then based on these questions, a set of metrics is defined to provide the quantitative information necessary to answer the questions.

The choice of GQM as a base for UFAPM was supported by the facts that it is the most widespread approach for software measurement plan definition, and, unlike the traditional hierarchical decomposition approach, it provides both an inference framework (top-down process) for deriving measures from organization or project goals; and a context for (bottom-up process) interpreting and analysing the collected measurement (Solingen & Berghout, 1999). GQM has also the benefit of providing traceability between what is being measured and the goals which allows for concentrated measurement operations on relevant and valuable metrics while removing measurement overhead (Dalton, 2019). Furthermore, it is rigorous, adaptable to many different environments (Achtaich et al., 2019; Calabrese et al., 2021; Darweesh et al., 2019; Yahya et al., 2017), and it has a gentle learning curve.

The Decision Support Framework for Metrics Selection (GQM/DSFMS) (Gencel et al., 2013) was also a source of inspiration, an extension of the GQM approach that was proposed originally to support software organizations to make decisions on metrics selection when planning measurement programs. GQM/DSFMS extends GQM by incorporating: decision-making mechanisms for metrics selection, a predefined Attributes/Metrics repository, and a traceability model. In this GQM extension, an intermediate abstraction level between question and metric levels was introduced. Instead of defining metrics directly from questions, the former are identified from a repository of Attributes/Metrics after the questions are mapped to their associated attributes. The repository contains a predefined set of standard and widely used metrics, classified according to

software attributes. This approach revealed its usefulness in metrics identification, reuse, and collection, and a cost saver of the measurement programme.

Figure II-1 depicts the UFAPM process. For each of the process phases, a set of guidelines is provided to give the meta-information required for its execution. Before proceeding with framework detailing, based on GQM guidelines, the main objective, which is defining an effective context-related agent properties measurement model, needs to be formulated using the GQM goal template proposed by Basili et al. (Basili et al., 1994). This first step will be the starting point of the UFAPM process: The Goal is to analyse **Agent** for the purpose of **characterisation (C)** with respect to its **properties,** in the context of the **Agent-Oriented (AO) Paradigm** from the point of view of **Agent-based System Developer.**

Phase 1: Goal Elicitation

The first phase is about refining the main goal into more simple ones; each sub-goal will represent a separate quality focus (agent property) under which the agent would be characterized.

In this phase, clear definitions of agent and agent properties need to be set, and a well-delimited measurement context and agent implementation-related details need to be settled. As guidelines for this phase, the following questions should be answered:

- What is an agent?
- What is the list of agent properties?
- What are the implementation details (programming language and implementation platform) and the measurement context of the agent under analysis?

The next sections will rely on the use of the following: the notation "**AODE (Li, Pi)"** to denote agent-oriented development environments with *Li* as agent programming language and *Pi* as agent developing platform; the two notions of "property" and "sub-goals" without distinction; and the property name to denote the sub-goals with this property as a quality focus.

The outputs of phase 1 are:

- A set of sub-goals "G_j": analyse **Agent** for purpose of **characterisation** (**C**) with respect to **the property 'j'** in the context of **AODE (L_i, P_i)** from the point of view of **Agent-based System Developer.**
- Definitions of the agent notions (agent and agent properties)

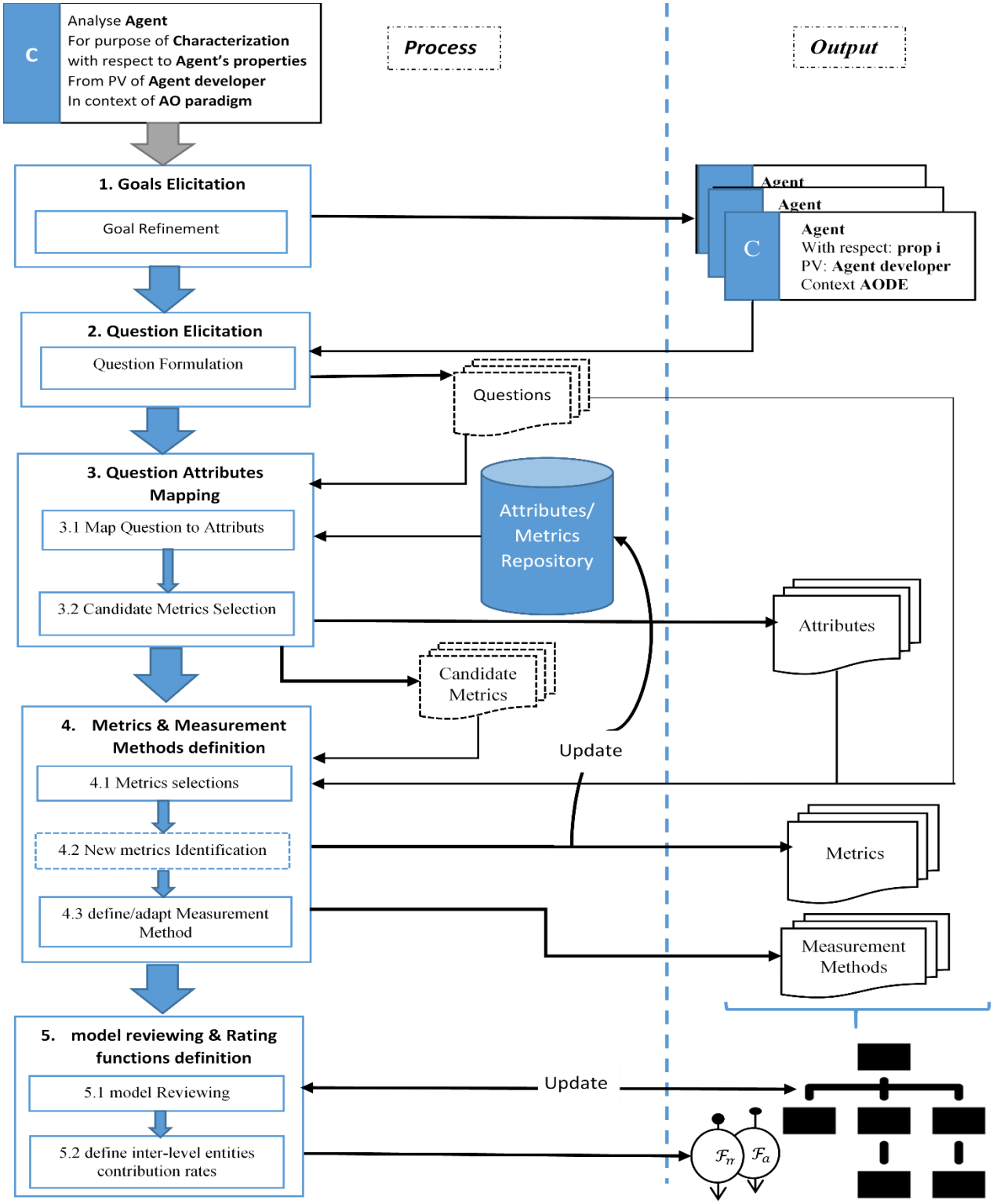


Figure II-1 The UFAPM Process

Phase 2: Questions Elicitation

Following the GQM approach, for each sub-goal identified previously, a set of questions that will help to characterize the sub-goal in a quantifiable way is specified. To ease the task of question formulation, a decision is taken upon the reuse of Basili's meta-questions (Basili et al., 1994) (after adapting them to the current context) to guide the questions elicitation; Three (3) groups of questions can be asked for each goal under study:

- How can the agent be characterised with respect to the current quality focus (property)?
- How can the agent attributes, that are relevant with respect to the actual property, be characterized?
- How would the relevant agent attributes be evaluated with respect to the actual property?

The outcomes of this phase are a set of questions that break down each sub-goal into its major components.

Phase 3: Question Attributes Mapping

To facilitate the task of metrics identification and to take advantage of previous works on agent measurements, an Attribute/Metrics repository was built from the results of the literature review. In this repository, each agent attribute is associated with a set of metrics able to measure it. The templates below (Table II-2 and Table II-3) were used to save in the repository the attributes and the metrics-related information respectively (currently, the repository is implemented within an Excel spreadsheet).

Item	Description
Attribute's id	A unique identifier for the attribute
Attribute's name	The attribute name
Attribute definition	A description of the attribute, and how it is related to the agent Properties
Set of Metrics' IDs	Set of metrics' IDs associated with this attribute

Table II-2 Attribute Template

Item	Description
Metric's id	A unique identifier for the metric
Metric name	The Metric name
Metric definition	A description explains what the metric is, what is being measured (the entity under measure), and how it is related to the agent attributes.
Measurement Method	Define how the metric can be computed.
Agent Implementation(s) details (Li, Pi)	Additional information from the literature review on the implementation contexts where the metric was used

Table II-3 Metric Template

The current phase consists of mapping the questions formulated in the previous phase to their relevant attributes within the repository to eventually identify the sets of candidate metrics that can be used to collect the quantitative information required to answer them. The abstraction reduction of questions into attributes brings more understanding and tractability into the final measurement model that is how the agent properties are related to its internal attributes and how they are measured.

The outcomes of this phase are:

- Breaking down the sub-goals into sets of measurable attributes.
- Identification of a set of candidate metrics that can be used to measure the identified attributes.

Phase 4: Metrics and Measurement Methods Definition

The aim here is to build a set of metrics with well-defined measurement methods, applicable to the current measurement context. This phase consists of four tasks:

- Firstly, a selection from the set of candidate metrics is considered, those that can be applied effectively to the measurement context (AODE (Li, Pi)), that is metrics with measurement methods that can be used within or adapted to, the current agent implementation context. Other selection criteria can be identified based on the needs of the framework's users. The following questions can help to identify other criteria:
    - Why does the agent need to be measured (to understand, to evaluate, to predict, etc.)?
    - When will the measurement process be executed (in the early development phases, in the end, etc.)?
    - What types of metrics are going to be used: subjective, objective (static, dynamic, or both)?
- Secondly, the fact that the Attribute/Metrics repository was populated from the results of different works (they do not necessarily share the same agent notions i.e. agent definition and implementation details), may raise some questions about the soundness of selected metrics, because attribute-metric relationship defined in one context can be not relevant in another. In this circumstance, the appropriateness of each metric is checked to ensure that it truly aids in quantifying the associated attribute in the current measurement context. The repository can be used to look for an eventual alternative metric(s).
- The third task is optional, it is about defining new metrics for attributes with zero (or only a few and insufficient) metrics. This step goes back to the original GQM approach where, based on the questions identified in phase 2 (especially the third group of questions), new metrics are proposed and added to the Attribute/Metrics repository. To accomplish this task, it is recommended to use the GQM/MEDEA (GQM/MEtric DEfinition Approach) (Briand et al., 2002) as a practical guide to designing sound and useful metrics.
- In the end, each metric's measurement method is reviewed. Driven by the agent measurement context (i.e. agent implementation details: agent programming language and the developing platform), the measurement methods are either defined (for the new

metrics), adapted (if necessary for the metrics selected from the repository) or just reused as they are (for metrics from the repository that can be reused as they are).

Phase 5: Models Reviewing and Contribution Functions Definition

At this phase, once all the previous ones are executed successfully, the measurement model is ready to be used to characterize the agent under study. However, two pitfalls were noticed from the first uses of the framework:

- Sometimes there is a *part-of* relationship between sub-goals i.e. a sub-goal (property) has as a subset of attributes, the same set that entirely defines another one.
- The mechanism of abstraction levels reduction from sub-goals (properties) to attributes to metrics eases the understanding of how agent properties are defined and related to its internal attributes, but it fails in explaining how the entities of inferior level (attributes or metrics) contribute to the associated entity of the superior level (property or attribute respectively).

To overcome these drawbacks, the following decisions were taken:

- Examining the resulting model to identify and model the relationships between the parts leads to a more compact model and would optimize the measurement process.
- Using goal importance rating or prioritisation techniques like Analytical Hierarchical Process (AHP) (Saaty, 1988), Multiple Attribute Utility Theory (MAUT) (Keeney & Raiffa, 1993), Cumulative Voting (CV) (Leffingwell & Widrig, 2000) or Hierarchical Cumulative Voting (HCV) (Berander & Jönsson, 2006) to model inter-level entities' contribution rates. The AHP was chosen as a rating technique since it allows checking the consistency of the adopted rating and here computing time is generally low (in this study) because the number of entities at each level is low (maximum 8 properties/attributes); for n properties, the number of comparisons needed with AHP technique is $n \times (n-1)/2$ (Saaty, 1988).

The results of this phase are:

- A more compact version of the measurement model where inter-properties (sub-goals) relationships (part-of) are modelled.
- Weighted sum functions that catch how the agent properties (resp. attribute) are characterized in a quantifiable way from the values of their associated attributes (resp. metrics)

4 | JADE's Agent measurement model definition

To both build confidence in the usefulness of UFAPM and to define the JADE properties agent measurement model required for the testability study of this thesis, UFAPM application into the context of the JADE agent is presented in this section.

Figure II-2, Table II-6 and Table II-7 illustrate the outcomes of this framework implementation. For brevity purposes, the metric level within Figure II-2 is not going to be shown since this level is implicitly represented within Table II-6 (the weighted sum functions associated with properties). Meanwhile, Table II-7 represents weight sum functions associated with attributes.

The starting point of the framework is the main goal formulation which is Analysing **Agent** for the purpose of **characterisation** with respect to **its properties** in the context of **Java agent development platform (JADE)** from the point of view of the **developer.**

The first phase: Goal elicitation:

Based on the predefined guidelines of this phase, the following definitions were adopted:

1. An agent is*: "a special software component that has an autonomy that provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own agenda … agents may interact both indirectly (by acting on the environment) and directly (via communication and negotiation). Agents may decide to cooperate for mutual benefit or may compete to serve their own interests*." (Bellifemine et al., 2007).
2. An agent may have as properties:
    - **Autonomy**: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state (Wooldridge & Jennings, 1995).
    - **Reactivity:** agents can perceive their environment, and respond in a timely fashion to changes that occur in it to satisfy their design objectives (Wooldridge, 2001).
    - **Pro-activeness** (Goal Oriented (S. Franklin & Graesser, 1997)) agents can exhibit goal-directed behaviour by taking the initiative to satisfy their design objectives (Wooldridge, 2001).
    - **Sociability:** agents are capable of interacting with other agents (and possibly humans) to satisfy their design objectives (Wooldridge, 2001).

- **Rationality:** the assumption that an agent will act to achieve its goals, and will not act in such a way as to prevent its goals from being achieved—at least in so far as its beliefs permit (Wooldridge & Jennings, 1995).
- **Intelligence**: an agent having the capabilities: of reactivity, pro-activity, and sociability (Wooldridge, 2001).
- **Granularity***:* degrees of Agent complexity (Dumke et al., 2009).

3. Agent-Oriented Development Environments (AODE): the agent under analysis is an agent developed on the JADE platform with Java as the developing language **AODE (Java, JADE)** or simply **AODE (JADE)**.

Based on these definitions, the main goal was refined into seven (7) sub-goals, each one representing a property (quality focus) under which the agent was analysed for characterisation.

Phase 2: Questions elicitation

Once the agent under characterisation is defined, a set of questions was formulated for each property identified previously. For the sake of brevity, not all the identified questions will be cited. Instead, only a subset of sociability-related questions is presented:

- What is the communication level of the agent?
- What are the interaction protocols that the agent uses?
- What is the cooperation level of the agent?
- What is the collaboration level of the agent?
- What is the negotiation level of the Agent?
- What are the types of FIPA[11] communicative act the agent can use in the interaction?
- What is the number of services the agent requests?
- What is the number of services the agent advertises?
- etc.

Intelligence property was the exception, no question was formulated because, in the adopted definition, intelligence is related to the reactivity, pro-activity, and sociability properties. Instead, this relationship was only modelled (the dashed arrows in Figure II-2).

Phase 3: Questions-Attributes Mapping

[11] http://www.fipa.org/specs/fipa00037/SC00037J.html#_Toc26729689

In this phase, questions are mapped to their relevant attributes from the repository. Figure II-2 (second decomposition level) shows the results of this mapping. For example, the previous sociability questions were mapped to three attributes: communication, cooperation and negotiation.

After mapping all the questions to their relevant attributes, all the metrics associated with these attributes in the repository were selected as candidate metrics for the next phase.

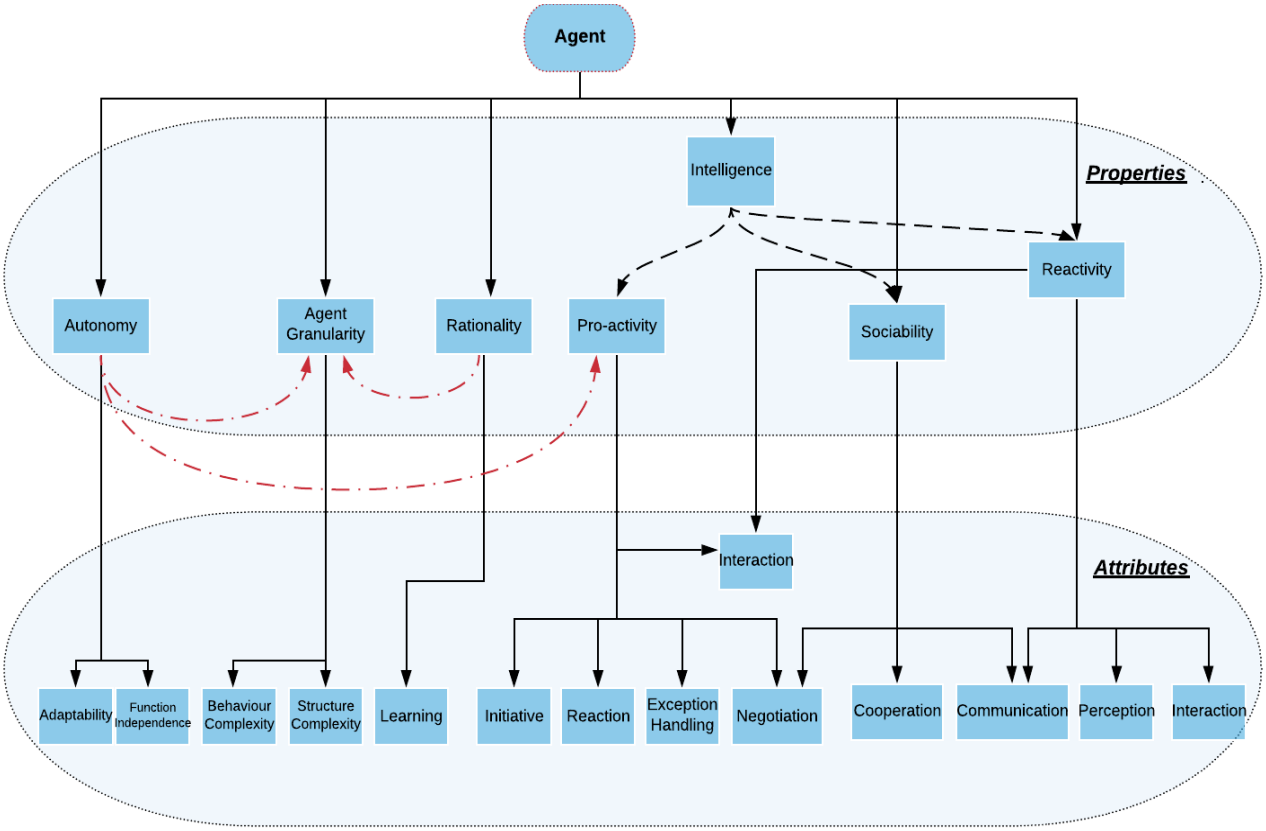


Figure II-2 JADE's Agent Measurement Model

Phase 4: Definition of Metrics and Measurement Methods

Following the guidelines of this phase, the checking of the applicability of each candidate metric was proceeded as a first step in aiming to select the best representative and suitable ones. The selection process was driven by the following criteria:

- The metric can be collected on JADE's agent.
- The measurement method is (or can be) implemented in Java.
- Only objective static metrics are selected; the choice to use only this metric type was supported by the facts:
    - Generally, measurements are needed in the early development phase where not all the agents are developed and ready to be run.

- Dynamic metrics are highly related to the execution scenario and sometimes (because of the dynamic nature of agent behaviour) multiple executions of the same scenario can result in different measurement values.
- This model will be used for static analysis of the JADE agent's code later as part of the JADE agent testability investigation (Chapter IV: 2.2 |)

Next, the soundness of each metric to the JADE context was investigated to ensure that each attribute-metric relationship is maintained, and the metric can truly aid in quantifying the associated attribute. It was found that some metrics associated with attributes: interaction, cooperation, perception and learning; were not appropriate due to context changes. For instance, *WMC* (weighted method per class) and *MsgIn* and *MsgOut* (number of messages received and sent respectively) were preselected to measure the interaction, but after reviewing their original context (where they were proposed) it appeared to be dominated by an OO paradigm, where agent services were implemented as public methods and service access and message exchange were implemented via external method calling. Meanwhile, JADE's agent developing philosophy encourages high encapsulation of agent functionality; all methods are supposed to be private, service registration or searching are managed via Yellow Pages service (provided by the Directory Facilitator agent) and all communications are explicit, via message exchange. To handle this inappropriateness, a decision to drop the *WMC* was made since another metric more representative of agent interaction was already preselected (rate of interaction code) and *MsgIn* and *msgOut* were replaced by *SerAdv* and *SerReq* (the number of services an agent advertises and requests; Table II-4).

As a third step, the completeness of each attribute's metrics set is checked, whether the identified metrics can provide the required quantitative information necessary to characterize the associated attribute. It was found that for most of the attributes, the repository provides the necessary metrics to evaluate them. The exceptions were within the attributes: cooperation, reaction, negotiation and function independence. In the case of cooperation and reaction, the selected metrics were not sufficient to quantitatively characterize these attributes. For instance, in the cooperation attribute, only the number of services an agent can advertise was selected as a metric, but this metric cannot capture alone the cooperation capability of an agent since an agent who advertises multiple services may have a selective collaboration process. Meanwhile, for the negotiation and function independence attributes none of the candidate metrics was selected because all metrics were either dynamic or subjective (Table II-1).

To handle the previous pitfalls, six (6) new metrics were defined, for instance, negotiation messages' rate (*NegoMsg*) (Formula. Eq.II-1) was used to quantify the level of agent negotiation. This metric is defined as the number of messages with a negotiation performative (Msg_{neg}) an agent

can exchange, divided by the number of negotiation performatives used in the JADE agent communication act (i.e. 4: CFP (call for proposal), propose, accept proposal, reject proposal)

$$NeoMsg = \frac{Msg_{neg}}{4} \qquad \textbf{Eq.II-1}$$

Once the set of metrics to use is known, each metric's measurement method is investigated to ensure its applicability to the JADE agents' code. It was found that for the 21 metrics identified only 05 metrics have measurement methods that can be reused as they are, most of these metrics are standard OO counting metrics **(**number of lines of code (LOC), number of methods (NbMtd), average number of methods parameters (AvgMtdPar), number of exception types (NbExp)). Meanwhile, seven (7) metrics need measurement methods adaptation since they were proposed in contexts different from the JADE context. For the rest of the nine (9) metrics, measurement methods had to be proposed, since six (6) of them are new ones and the other three (3) are from the repository but no measurement methods were identified in the literature. Table II-4 represents the final metrics adopted to evaluate the JADE's agent attributes. The column *source* indicates the origin of the metric, whether was selected from the repository (*Rep*) or newly defined (*New*); and the column measurement method (MM) indicates whether the method was adapted (A), defined (D) or reused (R).

Metrics	Title	Source (N, Rep)	Description	MM (R, D, A)
BeliefSize	Number of agent beliefs	Rep	Number of fields (global variables) defined within the agent class	A
NbBeliefAccess	Number of accesses (read, write) to agent's beliefs	Rep	Number of times the agent's fields were referenced in the agent code	A
NbBeliefUpdates	Number of updates of the agent's beliefs	Rep	Number of times agent's fields were updated (initialised or changed) in agent code.	A
NbExp	Number of exception types an agent can handle	Rep	Number of exception types catches in Agent code	R
SUC (KUG)	State Update capacity (average of beliefs updates)	Rep	Number of times the agent's field is updated, divided by the total number of agent's fields NbBeliefUpdates/BeliefSize.	R
Agree_Rate	Agreement Rate	New	Rate of agree messages an agent can send, divided by the total of agreeing and refusing messages an agent can send	D
Exec_Msg	Rate of executive Messages an agent sends	new	Number of executive messages sent is divided by the total of message types an agent exchanges. Executive message performative: informIf, informRef, QueryIf, QueryRef, Request, RequestWhen, RequestWhenever, subscribe, Proxy, Propagate	D
RecExec_Msg	Rate of executive messages an agent can handle	New	Number of executive messages an agent can handle (receive)	D
RecReq_Msg	Number of request messages an agent can handle	new	Number of Request messages an agent can receive. Request message performative: Request, Request-When, Subscribe, Request-Whenever, CFP.	D
NbSubBeh	Number of sub-Behaviours	New	Total number of agent's roles' sub-behaviours	D
NegoMsg	Rate of negotiation message exchanged	New	The Number of messages with negotiation performatives an agent exchange divided by 4	D

Metrics	Title	Source (N, Rep)	Description	MM (R, D, A)
AddedRole	Number of Roles an agent can play	Rep	Number of top-level behaviours an agent implements	A
RoleComplexity	Agent behaviour Complexity	Rep	Sum of Cyclomatic Complexity of Agent behaviours (formula Eq.II-4)	D
Adv_Ser	Number of services an agent advertise	Rep	Number of Agent services published in DF (the number of services passed as an argument to the DfService register() method)	D
Req_Ser	Number of services an agent Request	Rep	Number of services searched in DF (the number of services passed as an argument to the DfService search () method)	D
LOC	Number of Lines of code	Rep	Number of lines of agent's code	R
NOM	Number Of Methods	Rep	Total number of methods defined in the agent (including those within behaviours)	R
AvgMtdPar	Average of Method parameters	Rep	The average number of agent method parameters	R
SentMsg	Number of messages sent	Rep	Number of messages an agent can send (formula Eq.II-2)	A
RecMsg	Number of messages received	Rep	Number of messages an agent can receive (formula Eq.II-3)	A
LOC_COM	Rate of communication code	Rep	The rate of communication-oriented code in agent code	A

Table II-4 The Adopted Metric Set

Driven by knowledge of JADE agent programming language syntax and JADE API[12], a measurement-method adaptation process was conducted, where the following steps were considered:

- An ontology mapping: since some concepts are not defined explicitly in JADE's context namely:
  - Beliefs: beliefs are implemented as fields (global variables) defined within the agent main class (class extends **Agent** class).
  - Role: generally, the roles in JADE are implemented as behaviour (Bellifemine et al., 2007; Moraïtis et al., 2003; Massonet et al., 2002), thus a decision was made to consider each agent's top-level behaviour (not sub-behaviour) as a role implementation.
- A JADE API reviewing: to identify how some agent capabilities (attributes) may be implemented (communication, interaction, service advertising, or searching, etc.).
- Measurement method redefining: for the sake of handling JADE agent specificities.

Also, this process reveals that any measurement tool software must conduct code-mining tasks to identify:

[12] https://jade.tilab.com/doc/api/index.html

- The agent implementation code: in JADE, the agent code is scattered on multiple classes, all classes defining this agent should be identified.
- The portion of code implementing agent capabilities (communication, interaction, service request/advertise).

As an example of a measurement-method adaptation process, and for the sake of brevity, only three metrics are presented in this section: *SentMsg*, *RecMsg*, and *BehCcomplexity*.

a) Number Send/Receive Message*:*

In JADE, messages are sent and received either explicitly or implicitly. The former case is achieved by calling the methods: *send(), receive()* or *blockingReceive()*; whereas, the latter is within negotiation protocol implementation where the developer has to implement the appropriate methods responsible for preparing the message to be sent (**prepare...()**: *prepareCFP(), prepareRequestes(), PrepareResultNotoficatione(),* etc.); or processing the received message *(***handle…()**: *handleAgree(), handleInfome(), handleAllResponses(),* etc.). The formulas Eq.II-2 and Eq.II-3 represent *SentMsg* and *RecMsg* measurement function respectively.

$$SendMsg = SedExp + SendProto \qquad \textbf{Eq.II-2}$$

Where:

- *SendExp* is the number of *send ()* method call within agent code
- *SendProto* is the number of **prepare...()** or **handle…()** implemented methods that can trigger a message sending.

$$RecMsg = RecExp + RecProto \qquad \textbf{Eq.II-3}$$

Where:

- *RecExp* is the number of *receive()* and *blockingReceive()* methods call within agent code
- *RecProto* is the number of **handle…()** implemented methods that can process a received message.

b) Behaviour Complexity:

JADE's agent behaviour is explicitly defined by implementing a predefined class of behaviours. JADE has two main types of behaviours: simple behaviour and complex behaviour. In (Marir et al., 2014) a measurement method of a JADE behaviours complexity was proposed. The authors study only one type of agent behaviour (complex behaviour type). They represent this type as a graph (nodes are the composite behaviours and the transitions are the links between these composite behaviours) and they used the McCabe cyclomatic number (CC) (McCabe, 1976a) as

complexity metrics; meanwhile, they suppose the simple behaviour types to have complexity equal to 1.

The problem with this proposition is that it counts only the behaviour structure complexity but the inner complexity of both the simple behaviour's class and the composite behaviour's class (node in the graph) are not taken into account.

As fieldwork, a more general complexity measurement method proposed, the weighted method per class metric (*WMC*) (Chidamber & Kemerer, 1994), was used to count the inner complexity of both the simple behaviour and the composite behaviour. The JADE's agent behaviour complexity is the WMC_{ag} of agent main class (the class that extends **Agent** class) plus the behaviour complexity of all agent's behaviours Beh_iComp (Formula .Eq.II-4).

$$BehComp = WMC_{ag0} + \sum_i Beh_iComp \qquad \textbf{Eq.II-4}$$

Where:

- WMC_{ag0} is weighted method per class of the main agent class (class extends **Agent** class)
- i: is an agent behaviour

Beh_iComp is based on the type of behaviour as follows:

- Simple behaviour types (i.e. OneShotBehaviour, CyclicBehaviour, WakerBehaviour and TeakerBehaviour) is the *WMC* of that behaviour

$$BehiComp = WMC_{beh} \qquad \textbf{Eq.II-5}$$

- Complex Behaviour Types (i.e. SequentialBehaviour, FSMBehaviour, ParallelBehaviour):

$$BehComp_i = \left(\sum_j Beh_jComp\right) - Nb_{sb} + CCBeh_iSt \qquad \textbf{Eq.II-6}$$

Where:

- j: is a sub-behaviour (composite behaviour) of behaviour "i"
- Nb_{SB} is the number of sub behaviours
- $CCBeh_iStr$ is the cyclomatic number of behaviour "i" structure computed using the original CC formula (McCabe, 1976a) Formula.Eq.II-7.

$$CC = E - N + 2P \qquad \textbf{Eq.II-7}$$

  - *E* is the number of Edges or Transitions

- *N* is the number of Nodes or Sub-Behaviours
- *P* is the number of connected components or Parallel behaviour
  - In the case of *Sequential* or *FSMBehaviour* P=1, since there is only one behaviour (one connected component).

In the case of parallel behaviour, $BehComp_i$ is reduced to:

$$BehComp_i = \left(\sum_j Beh_j Com \right) \quad \textbf{Eq.II-8}$$

Since:

- E =0, no transition between sub-behaviours;
- P= number of parallel behaviour = number of sub-behaviours,
- $CCBeh_iStr = Nb_{SB}$

Phase 5: Model Reviewing and Contribution Rates Definition

In the end, the resulting model was reviewed to identify the part-of relationships, for instance, it was found that autonomy and rationality have, as a subset of attributes, the same set that quantifies agent granularity (behaviour complexity and structure complexity) that is to say that agent granularity is part of (or sub-properties of) both autonomy and rationality. Another case was between autonomy and pro-activity where the initiative, negotiation, interaction, reaction, and exception handling attributes of pro-activity are part of autonomy's attributes set.

Once these relationships are identified and modelled, in the final model (the red dash-dotted arrows in Figure II-2), the AHP method is executed to compute the contribution rates of every attribute (resp. metric) in the associated property (resp. attribute) value, and also the contribution rates of reactivity, pro-activity, and sociability in intelligence properties.

The AHP proposes a comparison fundamental scale of nine points (Table II-5) (Canco et al., 2021; Saaty, 1988). However, the comparison shed light only on a 5-point scale (1 to 5) since the levels from 7 to 9 are associated with decisions based on demonstrated claims and facts, which is not the case in this study. All related works in the literature on property-attributes or attribute-metrics association were based on subjective criteria. To the best of our knowledge, no controlled experiment was conducted on this question. Besides, the selected consistency rate threshold is 0.1 (that is CR must be <10%)(Pant et al., 2022; Saaty, 1990).

Intensity of importance	Definition	Explanation
1	Equal Importance	Two entities contribute equally to the objective
3	Moderate importance one over another	Experience and judgement strongly favour one entity over another
5	Essential or strong importance	Experience and judgement strongly favour one activity over another
7	Very strong importance	An entity is strongly favored, and its dominance demonstrated in practice
9	Extreme importance	The evidence favouring one entity over another is of the highest possible order of affirmation.
2,4,6,8	Intermediate values between the two adjacent judgements	When compromise is needed

Table II-5 Saaty's Comparison Fundamental Scale

All the AHP related calculation was realized by using **PriEsT**[13] **(**Priority Estimation Tool) (Siraj et al., 2015). Table II-6 and Table II-7 represents the results of this phase; column CR indicates the consistency rate values.

Properties	Weighted Sum Function F_a	CR
Granularity	0.75×Beh_Comp + 0.25×Str_Size	0
Sociability	0.4×Communication + 0.4×Cooperation + 0.2×Negotiation	0
Reactivity	0.309×Interaction + 0.582×Perception + 0.109×Communication	0.3%
Proactivity	0.449×Initiative + 0.123×Interaction + 0.072×Reaction + 0.123×Negotiation + 0.233×Exception handling	0.6%
Rationality	0.333×learning + 0.667×Agent_Granularity	0
Intelligence	0.40×Proactivity + 0.40×Reactivity + 0.20×Sociability	0
Autonomy	0.455×Agent_Granularity+ 0.263×Fct_Independece + 0.141×Proactivity + 0.141×Adaptability	0.4%

Table II-6 Properties Weighted Sum Function

Attribute	Weighted Sum Function F_m	CR
Perception	0.25×NbBeliefAccess + 0.75×NbBeliefUpdates	0
Adaptability	0.271×NbExp + 0.053×NbBeliefAccess + 0.122×SUC + 0.122×NbBeliefUpdates + 0.431×RoleComplexity	4.6%
Learning	0.2×NbBeliefAccess + 0.4×NbBeliefUpdates + 0.2×BeliefSize	0
Beh_Comp	0.206×Adv_Ser + 0.206×NsubBeh + 0.387×RoleComplexity +0.060×LOC + 0.098×NOM + 0.042×AvgMtdPar	4%
Negotiation	Nego_Msg	0
Str_Size	BeliefSize	0
Communication	0.5×SentMsg + 0.5×RecMsg	0
Cooperation	0.25×Adv_Ser + 0.75×Agree_Rate	0
Interaction	0.5×LOC_COM + 0.25×Adv_Ser + 0.25×Req_Ser	0
Initiative	0.667×AddedRole + 0.333×Exec_Msg	0
Reaction	0.167×Beh_Comp + 0.833×RecReq_Msg	0
Fct_Independence	1 - RecExec_Msg	0
Exception handling	NbExp	0

Table II-7 Attributes Weighted Sum Function

5 | JMP tool

To implement the results of the aforementioned framework application, and to evaluate their accuracy, a measurement tool for JADE's agent was developed named JMP (JADE Measurement

[13] https://sourceforge.net/projects/priority/

Project). The tool is a code analyser built upon the Eclipse *Java Development Tools* (JDT) project[14]. It can measure 25 agents' metrics.

The nature of JADE agent code (agent notion is scattered on multiple Java classes) imposes code mining to identify the agent (as an entity) code and then measures it. Driven by the idea that behaviours are the main agent building block in JADE (Bellifemine et al., 2007), the metrics' measurement methods were implemented to count metric values incrementally, all along the agent reconstruction process (Down-up):

- Counting simple behaviours metrics' values,
- Counting complex behaviours metrics' values by aggregating the values of its composite behaviours,
- Counting agent final metrics' values by aggregating the values of the agent main class, its behaviours and other related classes.

Figure II-3 illustrates the tool measurement collection process. Starting from a multi-agent-system project's source code, the inter-class dependencies are analysed using the CDA[15] (Class Dependency Analyzer) tool to define files (classes – Top-down) processing order (invert dependency order – Up-down). Then for each class, the AST (Abstract Syntax Tree) is produced. Following files processing order, the ASTs are visited to construct the agent and to execute metrics' measurement methods.

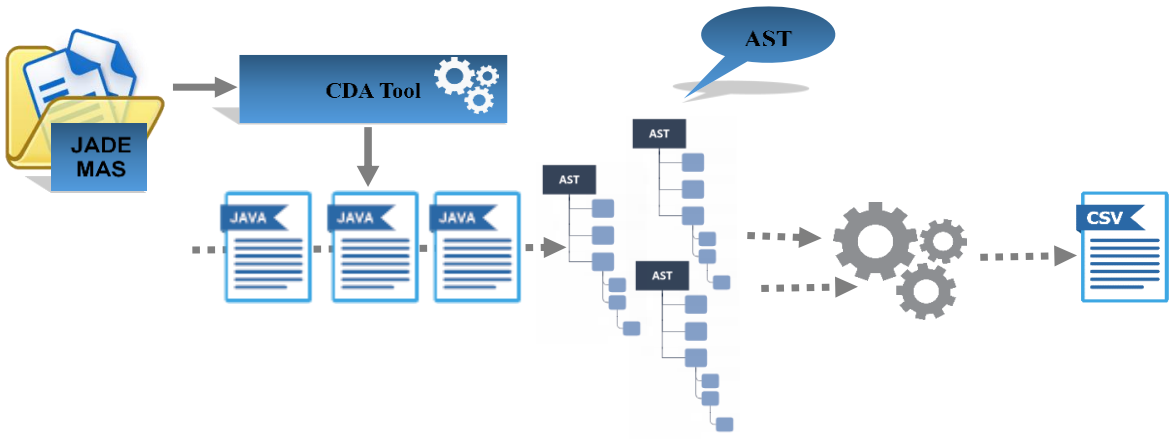


Figure II-3 JMP Measurement Collection Process

It is worth mentioning that to make the tool as loosely coupled as possible with measurement models, the JADE's agent model was not coded within the tool (i.e., tools do not implement the defined weighted sum functions to count properties final values). Instead, the tool generates only a

[14] https://www.eclipse.org/jdt/
[15] http://www.dependency-analyzer.org/

CSV file containing all the metrics' measurement values of agents. This file can be used later in any data analysis tool (Excel, SPSS, etc.) to count the final agent properties' values and eventually, conduct further data analysis if necessary. In the current study case, Pandas, a Python library (McKinney, 2010), was used to count agents' properties' final values. In the next section, more insights into this data post-process is given.

6 | JADE'S agent measurement model evaluation

The JMP tool was used on the 23 agents from seven open-source MASs (Chapter I: 6 |). Some basic statistics about these systems are presented in Table II-8. The table is a modified version of Table I-1, the behaviour complexity (formula Eq.II-4) is used instead of cyclomatic complexity. Systems 1, 2 and 7 are relatively small, meanwhile, systems 3 and 5 have medium sizes. However, system 7 is the biggest and the most complex one, with 9 agents with an overall behaviour complexity of 299.

MAS	System	Agents	LOC	Behaviour Complexity
1	Auction System -1-	2	322	41
2	Auction System -2-	3	397	44
3	Auction System -3-	3	525	69
4	Book Trading System	2	266	48
5	Hospital-Appointment Allocation System	2	520	99
6	Home Automation System	9	1972	299
7	Treasure Hunt System	2	254	34
	Total	**23**	**4256**	**634**

Table II-8 The MASs Understudy

The result of executing the JMP tool on these systems was a CSV file containing the metrics values of the 23 agents. The later data was post-processed with the Pandas package, a Python library. First, metric values were normalized (min-max normalisation) and then the measurement model's weighted functions were implemented to count the final agents' properties' values. Table II-9 represents the results of this process.

In the first system (Figure I-4), the measurement values show that both agents have a medium level in most of the properties. Both agents are autonomous, with no external influence on the auction execution process. They are intelligent; they try to get the best price (from their perspectives) for the job execution. However, the company agent has more granularity, reactivity, rationality and sociability. The reason is that it can participate in multiple negotiation sessions on different jobs at the same time and it can handle multiple coming bids for the same job. However, the carrier agent is more proactive than the company agent, it is implemented to take the initiative to underbid the other bidder, in the hope of forcing them to decline.

For, the second system (Figure I-5), which is like mentioned previously (Chapter I: 6 |), it is another implementation of the previous auction system; the splitting of the bargainer role between two agents had resulted in two simple agents with low levels of granularity, pro-activity, intelligence

and rationality. Even so, the auction agent has scored a high value of reactivity, a better value than the other agents have. The reason is that the auction agent is under the bargainer authority and its unique role is to negotiate with enterprises to get the best deal.

MAS	Agent	Granularity	Sociability	Reactivity	Proactivity	Intelligence	Rationallity	Autonomy
1	CarrierAgent	0.27	0.40	0.31	0.36	0.35	0.23	0.46
1	CompanyAgent	0.33	0.52	0.51	0.23	0.40	0.35	0.48
2	BargainerAgent	0.14	0.28	0.24	0.13	0.21	0.16	0.37
2	AuctionAgent	0.14	0.22	0.38	0.05	0.22	0.22	0.35
2	EnterpriseAgent	0.26	0.65	0.23	0.20	0.30	0.23	0.43
3	BidderHuman	0.34	0.35	0.30	0.30	0.31	0.31	0.49
3	Auctioneer	0.58	0.52	0.68	0.29	0.49	0.65	0.64
3	BidderComp	0.23	0.37	0.22	0.24	0.26	0.20	0.41
4	BookSellerAgent	0.17	0.38	0.25	0.23	0.27	0.15	0.39
4	BookBuyerAgent	0.22	0.42	0.23	0.12	0.22	0.19	0.42
5	HospitalAgent	0.35	0.62	0.40	0.45	0.46	0.31	0.35
5	PatientAgent	0.56	0.70	0.53	0.45	0.53	0.51	0.68
6	Controller	0.10	0.05	0.17	0.15	0.14	0.09	0.34
6	Room	0.49	0.42	0.42	0.33	0.39	0.42	0.43
6	Building	0.29	0.51	0.30	0.20	0.30	0.25	0.19
6	Demo	0.04	0.00	0.00	0.04	0.02	0.02	0.30
6	MeshNetGateway	0.34	0.55	0.22	0.41	0.36	0.25	0.26
6	Bulb	0.71	0.72	0.55	0.49	0.56	0.61	0.51
6	TempSensor	0.73	0.68	0.57	0.60	0.60	0.61	0.53
6	LightSensor	0.77	0.68	0.59	0.60	0.61	0.65	0.56
6	ToggleSwitch	0.61	0.63	0.52	0.47	0.52	0.52	0.45
7	Player	0.26	0.17	0.21	0.16	0.18	0.27	0.43
7	GameMaster	0.17	0.12	0.15	0.05	0.10	0.19	0.36

Table II-9 Agents Properties Values

Alternatively, the enterprise Agent has better granularity, intelligence, autonomy and sociability. It is implemented to interact with two different agent types (communicate with bargainers and negotiate with auction agents).

By comparing this system with the previous one, we see that the measurement values are consistent and follow the same trends. By considering that the buyer role in the second system is implemented on two agents instead of one in the first system, and by aggregating the values of the bargainer and auction, we obtain nearly the same property values as the company agent. In the same way, the carrier agent and the enterprise agent that have the same role (sellers), show the trends of the same values. The exception is within pro-activity, where the carrier agent is more proactive than the enterprise. The reason is that the carrier agent uses a more complex strategy to underbid the other bidders, whereas the enterprise agent only reduces the bid amount with a fixed rate (10%) at each time.

The third system[16] (Figure I-6) and from its measurement values, the auctioneer has high levels of granularity, autonomy, and rationality. It is more intelligent than the other agents. The auctioneer agent is the auction master. It is responsible for all auction events: star auction, receive bedding, announce the winner and handle payment transactions. Moreover, due to the simplicity of this auction type, the auctioneer is a reactive agent more than proactive (receives bids and announces the agent with the highest bid as the winner).

Otherwise, BidderComp and Bidderhuman are less complex. The BidderComp is a small agent with a simple bidding strategy (ALL-IN strategy), it has low levels of granularity, proactive and rationality. In contrast, BidderHuman has higher granularity than BidderComp, since it implements additional functionalities to handle the user commands. It has the same level of proactivity and reactivity because it is controlled by the user. It is more intelligent as it can interact with two different entities (user and agent). Both agents have the same level of sociability, they both implement the same communication schema.

Nevertheless, the measurement values of this system show some inconsistency when comparing the rationality and autonomy of BidderCom and BidderHummain agents. The latter scored more than the former, which is wrong since BidderComp is autonomous, contrary to BidderHuman, which is under user control.

After reviewing the agents' code and the proposed model, a pitfall was identified within autonomy measurement in the model. The problem is related to the number of executive messages received metric definition (*RecExeMsg*), used to measure the functional independence attribute. This metric counts only the message sent by agents; it does not consider the message via GUI. This pitfall is related to an early decision made about JADE's agent measurement. We judged no necessity (out of scoop) to measure agent GUI since in JADE the GUI is implemented in a separate class with a code that is one hundred per cent GUI related (i.e. a typical Java GUI code, no agent logic within) (Bellifemine et al., 2007).

Additionally, it was found that the rationality incoherent value is related to the way the BidderHuman was designed, the developer did not respect the principle of concern separation, as it is recommended with JADE agent's GUI development (Bellifemine et al., 2007). He integrated the functionalities of the user's input constancy verification (verify the bid and additional budget amounts) within the agent code and not within the GUI code, which impacted the learn-ability attribute measurement of BidderHuman (Table II-9).

[16] https://github.com/ardiyu07/jade-blind-auction

For the fourth system, (Figure I-7), its two simple trader agents (low level of granularity and rationality level), have a medium sociability level, and no negotiation mechanism is implemented as in the previous systems. Both agents scored nearly the same value in most of all properties, the difference between them is marginal (5% in most cases). The simplicity of the trade mechanism made these agents' behaviours more reactive than proactive. Both agents' autonomy levels are in the same range as previous systems (all these systems are implemented to find the best offer autonomously).

The measurement values for the fifth system (Figure I-8) show that the patient agent has better properties' values than the hospital agent, it is more complex and intelligent; the patient agent is implemented to be rational and autonomous in reaching its goal, that is trying to get the best preferable appointment possible, either by interacting with the hospital agent to get a better appointment or by swiping with other patient agents. The high level of patient agent autonomy is related to the complexity of the agent's goal. Meanwhile, the hospital agent is less complex (medium granularity). It is more proactive than reactive. It is implemented to manage appointments, handle patient requests and try to find an optimal solution for all patients' queries.

Compared to previous systems, the agents in this system have higher values of sociality than other ones; this is because, contrary to other agents, patient and hospital agents are the only agents where all the sociability capabilities (communication, collaboration, negotiation) are implemented.

Regarding, the home automation solution project, the sixth system (Figure I-9), the measurement values of its 9 agents show that the demo agent is a dummy agent. Its unique role is launching the system. The developers' choice was to implement this functionality within an agent, but it can also be implemented as a simple object, this is why most agent properties are null.

The controller agent is another dummy agent; it is a reactive agent with roles: display system state and transfer the user's command to the adequate agent. The actual implementation of this agent supports only the bulb manipulation, the reason behind showing low measurement values.

Building an agent is a simple communicator agent with good sociability, medium granularity, and low rationality and productivity. It offers room catalogue services for both rooms and the controller (room registration and room identification). It has a lower autonomy level since it depends on rooms' collaboration (room registration) to satisfy the controller's request.

Like the building agent, the room agent offers services of device catalogue that is device registration and identification, for devices' agents and the controller respectively. Besides, it can collaborate with the building agent to ensure room registration. The room agent shows the trends of the same values as the building agent but with higher values because it is more complex. It interacts with complex agents (device agents) and has more goals to satisfy (a third goal: room registration).

However, it has a sociality value lower than the building agent, although it communicates with three agents (instead of two, in the case of the building), since it is less collaborative than the building agent which automatically accepts any registration request; the room agent has a more selective process (based on device location).

Mesh Net getaway agent is a simple agent with a medium: granularity, rationality and intelligence. It interconnects the hardware devices (sensors/actuators) and their agents, which explains its good level of sociality. Mesh Net gateway is a proactive agent, it is responsible for setting up the connection with the mesh Net, managing devices' agent subscriptions for their devices' notifications, and translating commands from JADE ACL Messages into hardware commands and vice versa.

The temperature and the light sensors, the bulb and the toggle switch are devices' agents, they are the centre of this architecture, and they intermediate the user and device communications, which explains their high values in most of all properties' measurements. They have complex functionalities: register within the room and the mesh Net agent, get the user commands from the controller, transmit them to devices through the mesh Net gateway and inform the user of devices states via the controller agent.

Temperature and light sensors show better values than other device agents since unlike other agents they wrap active devices with a complex state (numerical values, instead of binary in the case of the bulb and the toggle switch). They interact with three agents: the room, the controller, and the mesh Net gateway, which explains their good level of sociability, pro-activity, reactivity and rationality. Unlike the bulb, communication between the sensor devices and the user is two-way, the sensor agents must translate user commands (numerical values) into analogical values understandable by the hardware device and vice versa, which is why they have high levels of granularity.

Meanwhile, the bulb agent warps a passive device (bulb) with a simple state (on or off) with no device notification to process which is why it has inferior values of granularity and pro-activity than sensors. However, unlike sensor agents, the bulb agent interacts with one more agent: the toggle switch agent, which can manipulate the bulb state; this is why the bulb agent is more reactive than proactive and has higher sociability than other devices.

Although the toggle switches agent warps passive devices with the simple state (on or off) like the bulb agent, it shows inferior values than it, since it has less functionality: register within the room and mesh net gateway, manipulate the bulb state by notifying the bulb agent on the physical hardware toggle switch state changes. Besides, the user cannot manipulate the toggle switch via the controller agent, which explains why it has a lower sociability value than other devices' agents.

The last system (Figure I-10), which is the smaller system, with two simple (most of their measurement values are low) reactive agents (low pro-activity and intelligence). The communication between the agents is just an asymmetrical, 2-agent communication, which explains the low level of agents' sociability. The master game agent is a dummy agent; it only responds to the player's moves by hints. Meanwhile, the player agent is more intelligent and rational; it is implemented to figure out the treasure location only by following the master game's hints.

7 | Threat to validity

Since this phase of the study is crucial and critical to the next ones, the threats to the validity of both the UFAPM framework and the defined model have been tried to be mitigated as much as possible

7.1 | The UFAPM framework

The framework was built upon the Goal-Question-Metric (GQM) approach, a well-known approach for measurement model definition, to minimise the risks related to the framework implementation and to guide the framework's user across different phases of model construction a set of well-defined guidelines were set up. However, all the process activities are done manually, which makes it tedious and error-prone. To mitigate this risk when the framework was used to define the JADE's agent measurement model, all these activities were double-checked.

Another threat is related to the current implementation of the attribute-metrics repository. Built from the agent measurements literature, this repository is proposed to ease the task of metrics definition. However, the repository is implemented in a spreadsheet file, this structure is sometimes hard to manage and does not allow full exposure of the attributes-metrics relationships.

Furthermore, the framework was validated on only one case study (JADE's agent), which can be seen as an external threat to validity. Further studies are necessary to handle this question, but intuitively there is no particular reason why the situation should be different for other agent definitions or other agent implementation platforms.

7.2 | The JADE agent properties measurement model

The main object of this study is to propose a solution for the internal validity threats that are generally associated with agent properties measurement model construction. These threats were well managed in the JADE agent measurement model construction. The UFAPM process permitted the identification of the full relationship between agent properties and metrics, to determine what attributes and metrics were missing or inconsistent, and to provide a context for interpreting the measurements after it was collected.

Additionally, the attributes-metrics repository helped to reduce the construction threats by providing a solid basis for attribute identification and metrics selection, since most of the attributes and metrics it contains have already been used and validated in the agent literature. However, restricting the type of metrics to use, to static metrics only, constrained the way some of JADE's agent dynamic attributes were measured such as negotiation, reaction, function independence, etc.

To evaluate the accuracy, the JADE's agent properties measurement model, 7 MAS were selected. The collected data shows that the defined model catches largely these systems' agents' properties. The main threat to this conclusion is the size of the sample. Although the sample was relatively small, it was built from different MAS solutions, with different sizes, ranging from small (treasure hunt system) to medium (home automation system), and with a variety of agent types (reactive, proactive, autonomous, subordinate, etc.).

Another limitation is identified with the autonomy measurement of agents with a GUI, specifically that the executive messages from the agent's GUI were not taken into account. This oversight was due to an early decision regarding JADE's agent measurement, where it was deemed unnecessary to measure the agent's GUI since, in JADE, the GUI is implemented in a separate class (a typical Java GUI class). This limitation is considered minor and does not undermine the obtained results nor the validity of the model, especially since it concerns only a specific behaviour for one agent. Nonetheless, it will be addressed in future work.

8 | Conclusions

Since the adoption of a complete and comprehensive measurement model for JADE agent is crucial for the main study of this theses, a specialized literature review was performed on agent properties measurement to construct a body of knowledge on this matter. It was found that even though multiple studies have been undertaken, they are incomplete. Only a subset of properties was studied each time. The lack of standard definitions of agent and agent properties makes the unification of these works very tedious. Furthermore, the hierarchical decomposition approach, used generally for properties measurement model specification, fails to explain how the properties decomposition process was undertaken and whether the adopted metrics are sufficient and complete.

Aiming to overcome these drawbacks and reduce the internal threat of the main research on JADE agent testability investigation, in this chapter a framework for agent properties measurement built upon the Goal-Question-Metric approach is defined. Starting from an agent definition, agent property list, and well-delimited measurement context; a tailored agent measurement model is defined by refining the agent properties into attributes, which are measured by a set of well-defined, sound and complete metrics. The framework proposes guidelines to support the model construction process, and a repository of Attributes/Metrics gathered from the literature review.

The framework was applied to JADE's agent to define a complete and well-detailed model for JADE's agent properties measurement, where JADE's agent properties are related to its internal attributes and measured using static metrics; the definition of new metrics for agent measurement; and the proposition of new measurement methods for some metrics identified within the literature review.

Additionally, JADE measurement project (JMP) tool was developed. The tool that supports the defined model. Its application was demonstrated on the adopted set of open-source MAS. It was shown that our measurement model could largely measure JADE's agent properties.

Chapter III: A Testing Framework for JADE Agent-Based Software

1 | Introduction

In this chapter, the focus is on JADE's agents testing. The specialised literature reviews and searches of open-code repositories, for both high-quality JADE agent testing frameworks and test cases that can serve as a base for the testability investigation revealed that existing testing solutions are weak and unreliable to some extent. They primarily focus on interaction testing and debugging of exchanged ACL messages (Gómez-Sanz et al., 2009; Coelho et al., 2007) .

For the unit level testing, it is widely accepted that the JUnit framework can be utilised to test agents' units (Coelho et al., 2007), as JADE agents are essentially Java code. But, the particularities of JADE's agent code render the exclusive usage of JUnit challenging and sometimes impractical. JADE's programming principles emphasize protecting the internal logic (behaviours) and knowledge (beliefs) of agents from external access and modification (Bellifemine et al., 2007). Also, the JADE agent is more than just an object; it is a complex entity. The code is distributed across multiple classes and relies on the platform itself at runtime, making effective unit testing impossible without taking these characteristics into account. To the best of our knowledge, no viable solution for efficient testing of JADE agents at both unit and interaction levels exists.

Moreover, the identified test cases were simple and were developed, in most of the cases, without any clear testing strategy, just as usability showcases of the proposed solutions. Thus, highly diverse and cannot be to serve as a foundation for the main testability research.

To overcome these issues and to provide solid ground for the next step in the testability study, a complete and effective testing framework for JADE's agent had to be developed and used to create test cases for systems under study.

In the coming sections, JTF (JADE Testing Framework) (Kalache et al., 2023) is presented. It allows testing of any agent on both the unit level via a component named UJade, built upon JUnit and PowerMockito; and on the agent level via the JAT4 component, an enhanced version of JAT (JADE Agent Testing) framework (Coelho et al., 2007), a well-known framework for JADE's agent interaction testing.

The chapter is outlined as follows: Section 2 presents a review of existing solutions, while section 3 outlines the defined requirements behind building JTF. Section 4 details the JTF architecture and describes how test cases can be implemented using it. To demonstrate the usability and effectiveness of JTF, an empirical study based on mutation testing was conducted on the seven selected MAS. The methodology, results, and validity threats of this experiment are discussed in Sections 5, 6, and 7, respectively. Finally, Section 8 presents the conclusions.

2 | State of Art

The initial efforts in testing JADE agents are encapsulated in the JADE Test Suite[17], proposed by JADE's development team (Caire et al., 2004). This framework was designed to facilitate the construction of test suites effortlessly, cheaply and incrementally. It employs a two-tier testing model: at the first level (unit), the atomic capabilities of an agent, such as message creation, sending, receiving, and responding to incoming requests, are tested. Then, at the second level (agent), the functionality (a collection of capabilities) of the agent. Subsequently, the framework was enhanced within the context of the PASSI methodology (Chella et al., 2004; Massimo Cossentino, 2005). The integration of the Agent Factory tool (Cossentino et al., 2003) enabled the generation of mock agents to simulate the environment of the Agent Under Test (AUT). Yet, the JADE Test Suite is inadequately documented, with the existing documentation (Cortese et al., 2005) being incomplete and lacking detailed information on test case coding. Additionally, no practical use cases of the tool are available.

In subsequent developments, Coelho et al. (2006, 2007) introduced the JADE Agent Testing (JAT) framework for unit testing JADE-based multi-agent systems. Although termed unit testing, JAT focuses on testing agent interactions. JAT employs a role-based approach for defining test cases, wherein a mock agent is defined to test each role of the AUT. A JAT mock agent is a fake implementation of a real agent, executing a test script that stimulates the AUT by sending messages and verifying the conformance of the responses. Furthermore, to simplify the implementation of mock agents, JAT offers a template-based mechanism that can automatically generate mock agent code from an XML specification of an interaction protocol.

Next, an adapted version of JAT for BDI-agents, termed JAT4BDI, was introduced (Cunha et al., 2015). This new iteration enhanced JAT's monitoring capabilities to observe the agent's reasoning cycle during interactions. In addition to verifying agent communication, assertions regarding the agent's goals, beliefs, and plans can be formulated. Both JAT and JAT4BDI employ a black box testing strategy, focusing solely on agent interactions and their effects on the agent's mental state (goals, beliefs, etc.), while leaving the agent's internal structure and logic unexamined.

As a part of the agent-oriented development methodology (AODM) Tropos, Nguyen et al. (2008) proposed a testing tool, named ECAT, which comprises three components: a test suite editor that semi-automatically generates test suite skeletons from XML specifications, a Tester agent and a Monitoring agent to monitor and debug communications and events among agents. Nguyen et al. used a goal-oriented approach for defining test cases for JADE agents. For each goal of an AUT, a

[17] https://jade.tilab.com/download/add-ons/

test suite is defined to verify goal fulfilment. The Tester agent executes the test suite by stimulating the AUT with messages and verifies the correctness of its responses.

Similarly, in the aforementioned work, Nguyen et al. employed a black box testing strategy, relying on agent sociability for testing. Nonetheless, it is conceivable that some goal accomplishment strategies of the agent (behaviours) exclude communicative actions (e.g., inner goals), hence preventing the Tester agent from validating them. Moreover, attaining a goal does not inherently signify that the strategy was implemented accurately and without of mistakes.

Gómez-Sanz et al. (2009) proposed an approach within the INGENIAS Methodology for debugging and testing JADE agent interactions. For debugging, they utilise the ACLAnalyser tool to track and log individual agent conversations. The logs generated can be used to verify the correctness of interaction sequences, detect overhead data exchanged between agents, and identify unbalanced execution configurations. Additionally, these logs can support advanced analyses, such as applying data mining techniques to uncover emergent patterns at the system (social) level. However, for testing, a framework was proposed, built upon JUnit and integrated within the INGENIAS Development Kit. It focuses on verifying the effects of interactions on the agent's mental state and uses a model-driven approach for generating test suite skeletons.

Gómez-Sanz et al. solution has limitations: The ACLAnalyser primarily visualizes inter-communication relationships rather than verifying them, and it does not check the conformance of exchanged messages (e.g., message content, performative, etc.). Furthermore, like previous works, the proposed test framework centres on the effects of interactions and does not consider the agent's behaviour.

Carrera et al. (2014) introduced the BEAST Methodology, an agile agent testing approach based on the Behaviour Driven Development (BDD) paradigm (North, 2006). The proposed BEAST software Tool[18] automates the generation of JUnit test case skeletons from the BDD scenario specifications. It employs a black box testing strategy that relies on mocks. The mocks are either mocks of ordinary Java objects or one of the three provided mock-agents: ResponderMockAgent, ListenerMockAgent, and MediatorMockAgent. Additionally, the tool provides utility methods to access and manipulate both the JADE platform and the Agent Under Test (AUT), facilitating the setup of test fixtures and the verification of AUT interactions and its mental state.

Additionally, to limit tests to agent interactions and the evolution of its mental state, the BEAST tool supports only three types of mocks. Unlike JAT's mocks, where developers can define mock behaviours to implement complex interaction schemas (such as protocols), BEAST's mocks

[18] http://github.com/gsi-upm/BeastTool/

are treated as black boxes. Only their communication methods can be stubbed to set the messages to be sent or received.

To summarize, the existing JADE agent testing solutions predominantly emphasise agent-level testing, specifically testing agent interactions and their effects on the agent's mental state, while neglecting unit testing of individual agent's components. These limitations are evident when examining JADE-based MAS in code repositories such as GitHub, GitLab, SourceForge, and Google Code Archive. Few projects provide agent test cases, and even fewer include unit tests for agents. In the rare instances where unit testing is implemented, developers often resort to an object-oriented approach using JUnit, which necessitates making agent methods public—contrary to agent programming principles (Bellifemine et al., 2007).

3 | Requirement for a Complete Jade Agent Testing Framework

To address the existing solutions limitations a new comprehensive, effective, and user-friendly framework for testing JADE agents needs to be designed. The framework should:

- **Made no assumption on agent code**: from either a JADE agent code or a specification, a developer can use the framework to write and run functional tests.
- **Provide core functionality of a testing framework**: This includes the ability to write, run test cases, and report test results.
- **Permits to test agents at unit and/or agent levels**: at the former, the agent is seen as a white box, with each component scrutinised to verify its corrected functioning. At the agent level and from a black box point of view, the agent's capacity to achieve its objectives and to interact with its environment is tested.
- **Permits to spy on agents while they are running on the platform**: JADE's API promotes high abstraction, allowing developers to focus on the inner logic of agent behaviours while hiding aspects related to behaviour management and agent state management. Since an agent's behaviours can only be tested during runtime, it is essential to dynamically spy, monitor, and stub the agent's internal actions.
- **Provide utilities to access and manipulate an agent's inner state**: JADE's programming principles advocate for protecting an agent's inner state (mental state and inner behaviour) from external access, implementing them as private entities (Bellifemine et al., 2007a). The framework must facilitate the manipulation and verification of agent's inner elements, like methods, behaviours (inner logic), and field (beliefs and mental state).

- **Provide utilities to bring the agent into a desired testing state/behaviour:** an agent may exhibit different behaviours and execution states, requiring the ability to bring it to a desired initial state before testing it.

- **Provide utilities to mock agent's environment**: agent is both an interactive and reactive entity it can affects and/or be affected by its environment (agents, objects, and other systems, etc.). Thus, the need to Mock agent's environment to set the test fixture.

4 | JTF presentation

4.1 | JTF's architecture

Driven by the previous requirements JADE Testing Framework (JTF) was developed . It offers a straightforward, comprehensive, and efficient solution for JADE's agent testing. It comprises three components: UJade for agent's unit testing, JAT4 for agent interaction testing and a domain-specific language (DSL) that provides a set of adopted assertions for JADE's agent testing. The framework architecture is presented in Figure III-3 . JTF is built upon JUnit Mockito and PowerMockito[19].

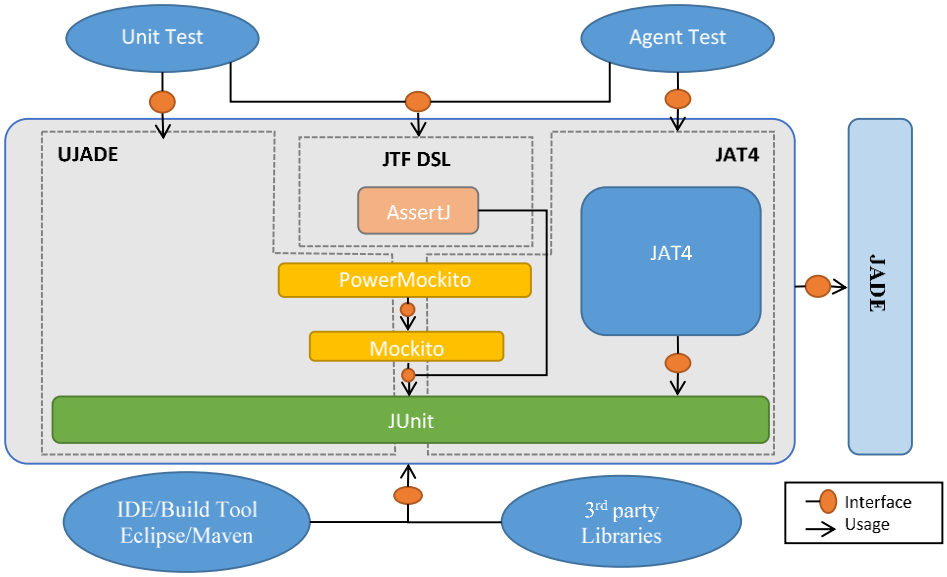


Figure III-1: JTF General Architecture

Given that JADE is a Java framework, it is logical to present JUnit as the foundation for framework. JUnit is the most prevalent and robust Java testing framework, facilitating a reduction in the JTF learning curve, as JTF tests, both unit and agent level, are fundamentally JUnit tests. Besides, it facilitates the integration of JTF with IDEs and building tools such as Eclipse and Maven. Finally, most of the JUnit ecosystem libraries remain compatible with JTF; JaCoCo for code coverage and the PITest library for mutation testing, are examples of such libraries.

[19] JUnit4(v4.13.1), powermockito2 (v2.09) and Mockito3 (v3.3)

PowerMockito is an expansion of the Mockito library, which is regarded as one of the premier Java mocking libraries. PowerMockito augments Mockito by enabling the access, manipulation, and mocking of private and final methods within Java classes, which is particularly relevant in this situation since all JADE agent's fields, methods and behaviours are private.

A typical use case (Figure III-2) of JTF starts with a test specification which may consist of either an individual test case or a test suite, formulated following the JUnit syntax, a developer via either an IDE or a building tool can run the test(s). JTF employs the PowerMockito test runner to execute tests and provide the test report. The report is a conventional JUnit report that specifies the outcome of each test, whether it passed or failed, along with an explanation for eventual failures details (the location and reasons).

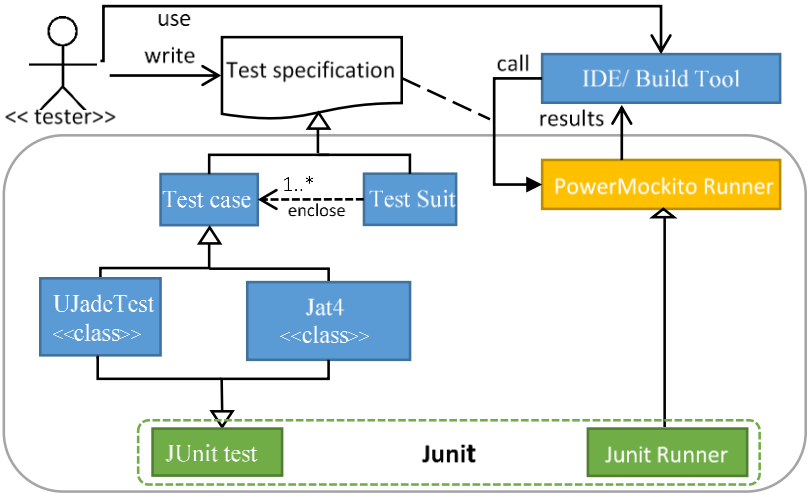


Figure III-2: JTF Use Case

Prior to elaborating on JTF inner architecture and tests specifications, it is pertinent to note that the subsequent sections utilise the book trading system (Caire, 2009) a case study.

4.1.1 | UJade

The core concept of UJade (package: *jtf.level.unit* - Figure III-3), is to utilise PowerMockito capabilities to create a spied version of the agent under test (AUT). This spied agent operates like any other JADE agent, but unlike the original agent, it is possible to fully access, manipulate, monitor and verify its internal actions (behaviours) and mental state. UJade provide, all-necessaire functions that allow to easily and effectively test agent units (sub-package: *jtf.level.unit.mockito* Figure III-3), like functions to

- Access and set (resp. call) agent's classes' fields (resp. methods).
- Access and manipulate agent's behaviours that are implemented as inner Java classes or anonymous ones.

- Stub some agent's general methods (like message reception, behaviour adding, etc.).
- Run in isolation and captor the return values of agent's and behaviours' methods.
- Block the execution of undesirable behaviours along with the possibility of spying on the desired one (UJade provide a special stubbing function for agent's **addbehaviour()** method).
- Captor and verify agent inner state (resp. methods or behaviour) change (resp. call or execution).
- Mock JADE platform services, like DF service.
- …etc.

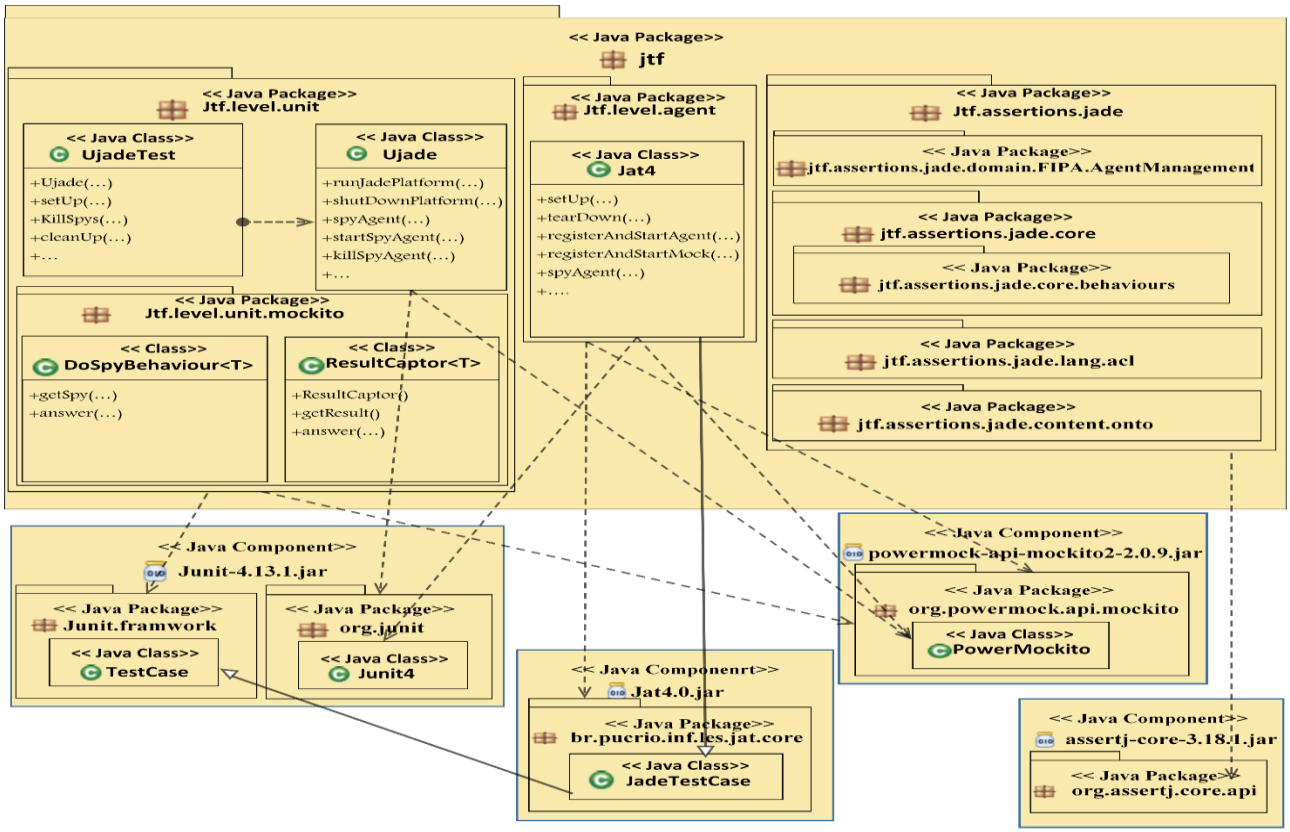


Figure III-3: JTF Package Diagram

Figure III-4, illustrates a sequence diagram for a typical agent unit test, focusing on testing agent behaviour and mental state.

- **Spy and Setup**: This fragment shows how UJade uses PowerMockito to create a spied version of the agent. The agent's internal actions and state are stubbed and modified to bring it to a desired testing state.

- **Mock AUT's Environment**: In this fragment, environmental objects and systems that the agent under test (AUT) depends on or is coupled with are mocked using Mockito and PowerMockito.
- **Run AUT**: Once the agent under test is executed, UJade employs PowerMockito's verification functionality to monitor and verify the agent's internal actions.
- **Assert AUT Inner State**: Finally, the agent's mental state is asserted to verify its evolution.

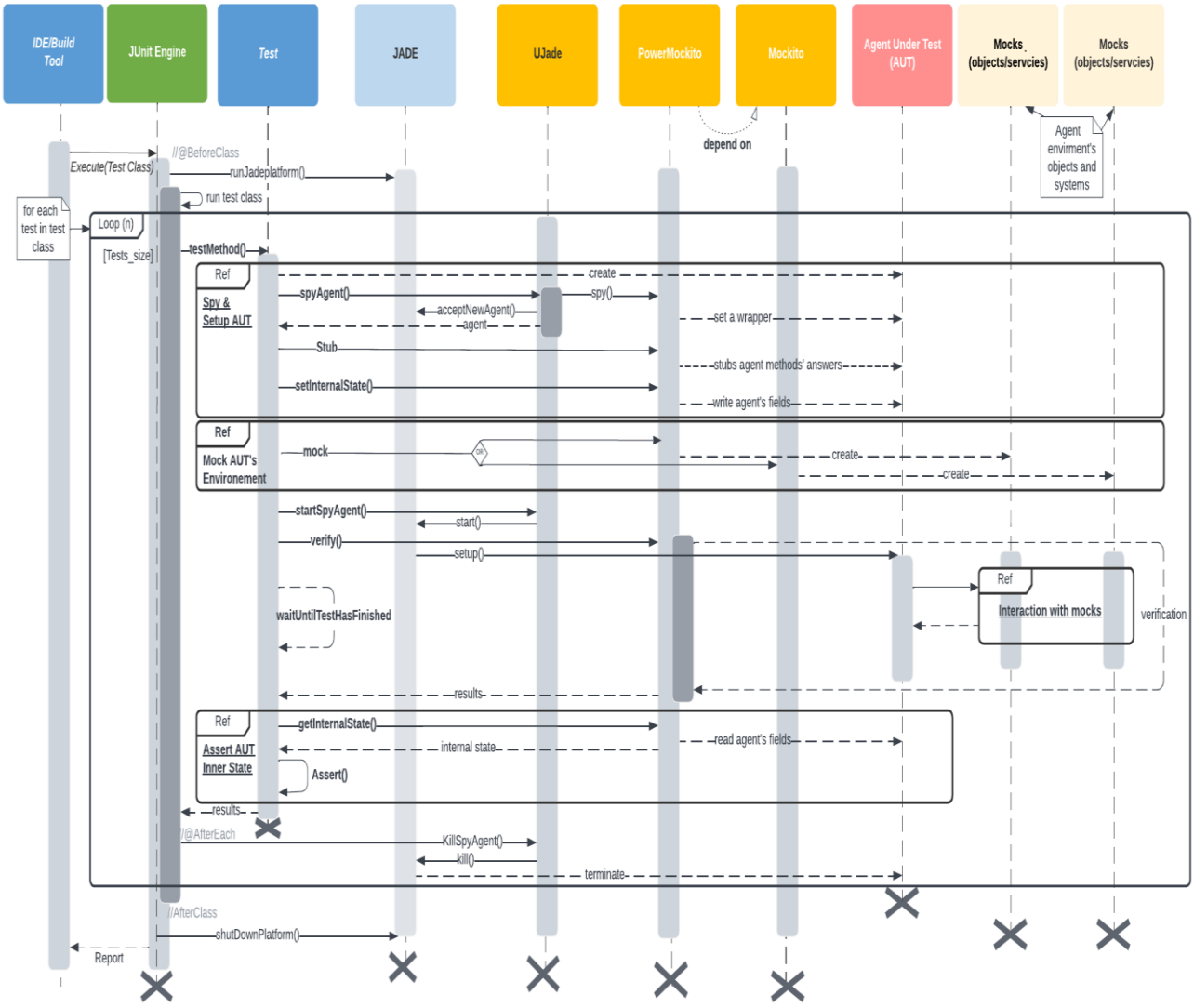


Figure III-4: Sequence diagram for a typical JTF unit test

4.1.2 | JAT4

For Agent level testing, JTF incorporated an enhanced version of JAT farmwork (Coelho et al., 2007), named JAT4, which integrates also PowerMockito and adopts the JUnit4 syntax. With JAT4, writing concise, interaction-focused tests is much simpler. Unlike the original JAT, where the mock agent had to handle priory all the interaction sequences necessary to transit the agent under test from its initial state to the desired one, JAT4 like UJADE, leverages PowerMockito, to directly

manipulate the AUT's internal state and to stub and mock, if necessary the AUT's internal actions and/or JADE platform services.

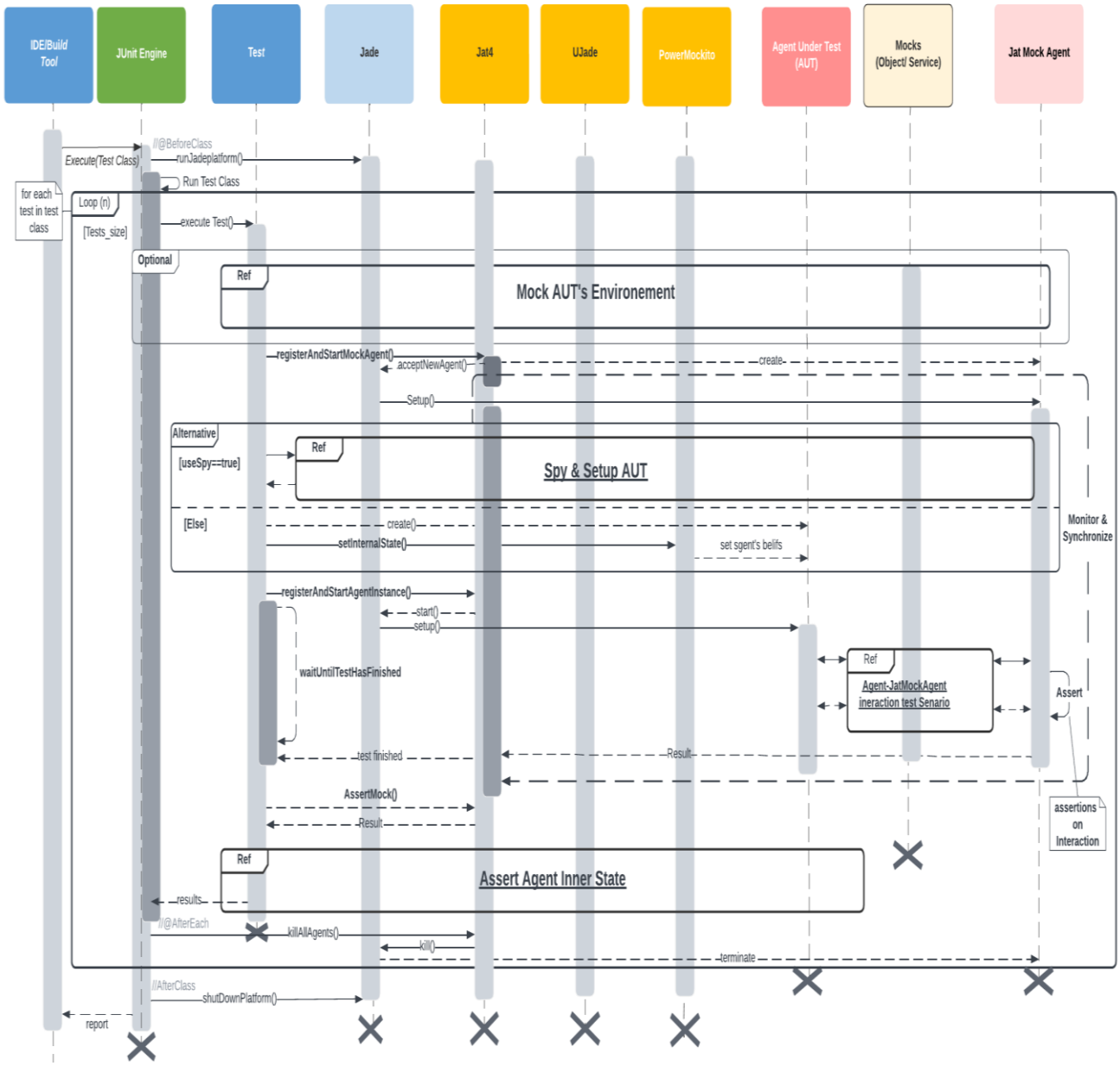


Figure III-5: Sequence diagram for a typical JAT 4 test

Furthermore, JAT4 is compatible with all JADE platform versions 3.x and 4.x, and it is available as a separate project (Figure III-3, JAT4.jar) to facilitate future extensions of JTF, such as the inclusion the another version for JAT related to JADE's BDI agents (JAT4BDI).

The sequence diagram in Figure III-5 illustrates a typical JAT4 test case:

- **Create Mock Agents:** The test begins by creating one or more JAT mock agents that will interact with the agent under test (AUT). These mock agents are crucial for implementing the testing scenario and evaluating the AUT's responses.

- **Mock AUT's Environment**: Optionally, PowerMockito can be used to mock non-agent environmental objects, like unit-level testing.
- **Run Spied or Real Agent**: Depending on the testing scenario, JAT4 can run a spied version of the agent with stubbed actions and a predefined mental state (to set it in a desired state) or the real agent in its initial state.
- **Synchronize and Monitor**: JAT synchronizes and monitors both the mock agents and the AUT.
- **Notify Results**: Once the mock agents complete their tasks, they notify JAT of the results.
- **Assert Mental State**: Finally, the agent's mental state is asserted to verify its evolution, like unit-level testing.

4.1.3 | JTF's DSL

JTF is built on JUnit, a widely used testing framework, but it incorporates a domain-specific language (DSL) to provide a more expressive and suitable syntax for testing JADE agents. The chosen DSL is based on the AssertJ library[20] (Figure III-3, package*: jtf.assertion.jade*), which offers a comprehensive set of assertions. These assertions not only simplify test writing but also enhance the readability and understandability of the test code, resulting in improved code quality compared to basic JUnit assertions (Leotta et al., 2018). The decision to use AssertJ over other assertion libraries like Hamcrest[21], enhances tester productivity (Leotta et al., 2019, 2020).

For example, in Figure III-6, there is a mock buyer behaviour code designed to test the seller agent's order processing at the agent level. In the lines between 17 and 33, the mock buyer checks if the seller is responding to its book order with the correct message format. Without the proposed DSL, five assertions (commented lines 26 to 30) would be necessary to verify the reply message format. However, using JTF's DSL allows for this check to be performed with just one assertion (line 33).

Overall JTF DSL includes five sets of assertions (Figure III-3, package*: jtf.assertions.jade*):

- Assertions related to agent definitions (e.g., AID, launching arguments, state).
- Assertions to verify the structure of an agent's composite behaviours.
- Assertions for agent and service descriptions in the DF catalogues.

[20] https://assertj.github.io/doc/
[21] http://hamcrest.org/JavaHamcrest/

- Assertions for ACL messages.
- Assertions for ontology verification.

```
12 @Role("PurchaseOrdersServer")
13 public class Send_PurchaseOrder_Mock_Beh extends OneShotBehaviour {
14    @Override
15    public void action() {
16       try {
17           ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
18           order.addReceiver(new AID("Seller2", AID.ISLOCALNAME));
19           order.setConversationId("book-trade");
20           order.setContent("The Hobit");
21           order.setReplyWith("order" + System.currentTimeMillis());
22           myAgent.send(order);
23           // case 1
24           ACLMessage reply = myAgent.blockingReceive(5000);
25           /*
26            * assertNotNull(reply); assertEquals(order.getLanguage(),reply.getLanguage
27            * assertEquals(order.getOntology(), reply.getOntology());
28            * assertEquals(order.getProtocol(),reply.getProtocol());
29            * assertEquals(order.getConversationId(), reply.getConversationId());
30            * assertEquals(order.getReplyWith(), reply.getInReplyTo());
31            */
32           // JTF_DSL
33           ACLMessageAssert.assertThat(reply).isReplayTo(order);
34           assertEquals(ACLMessage.INFORM, reply.getPerformative());
35
36           order.setContent("MoonLess");
37           order.setReplyWith("order" + System.currentTimeMillis());
38           myAgent.send(order);
39           // case 2
40
41           reply = myAgent.blockingReceive(5000);
42           // AssertJ
43           ACLMessageAssert.assertThat(reply).isReplayTo(order);
44           // assertEquals("book-trade", reply.getConversationId());
45           // assertEquals(order.getReplyWith(), reply.getInReplyTo());
46           assertEquals(ACLMessage.FAILURE, reply.getPerformative());
47           assertEquals("not-available", reply.getContent());
48           ((JadeMockAgent) myAgent).setTestResult("OK");
49
50           } catch (AssertionError | NullPointerException e) {
51           e.printStackTrace();
52           ((JadeMockAgent) myAgent).setTestResult(e.getMessage() == null ?
53                     "" + (e.getStackTrace()[0].getLineNumber()) : e.getMessage());
54           }
55    }
56 }
```

Figure III-6: Mock-Buyer's Behaviour

4.2 | Test cases implementations on JTF

JTF is based on JUnit 4, thus test cases' definitions and test methods implementation follow JUnit4 syntax. To define a test case, a tester must extend one of these two classes: UjadeTest.Java for a unit-level test case or JAT4.Java for an agent-level test case. These classes contain all behind scene code, necessary for calling the adequate JUnit test runner, preparing (resp. cleaning) the test environment before (resp. after) test case running, for example: lunching the jade platform; cleaning the platform and killing agents after each test execution; shutting-down the platform after test case ends, etc.

Figure III-7 and Figure III-8 illustrate the skeletal structures of the test code for JTF's unit-level and agent-level test methods respectively. Figure III-9 illustrates the typical code skeletons of a JAT's mock agent's behaviour (**action()** method). The bold lines denote the main instructions, while the others are optional, contingent upon the test scenario.

```
public void test_name() {
     // Mock or spy on objects (environment objects, DFService or other AUT's beliefs)
     // Stub mocks' or spies' methods (to set predefined answer).
     // Spy on AUT
     // Bring AUT to the desired state by:
          // Setting AUT's beliefs.
          // Stub AUT's inner-actions (adding behaviour, message reception...)
     // Start AUT or call agent's or behaviour's method under Test
     // Wait for AUT execution.
     // verify AUT's inner-actions
     // Assert agent beliefs state
}
```

Figure III-7: a unit-level test case code skeleton

```
public void test_name() {
     // Mock or spy on objects (environment objects, or The DFService ).
     // Stub mocks' or spies' methods (to set predefined answer).
     // Spy on AUT (needed in some cases to Bring AUT into desire state)
     // Bring AUT to the desired state by:
          // Setting AUT's beliefs.
          // Stub AUT's inner-actions (blocking adding unwanted behaviour)
     // Define and start JAT's mocks agents
     // Start AUT.
     // Wait for JAT's Mocks execution.
     // Assert that JAT's Mocks had ended correctly
     // Assert AUT beliefs' state
}
```

Figure III-8: An agent-level test case code skeleton

```
public void action() {
     try {
          // interaction implementation (sequence of send and receive messages)
          // Assert correctness and conformance of receipted messages
          // Set test as passed
     } catch (Exception e){
          // Set test as failed
}
```

Figure III-9: JAT mock agent's behaviour's action method

As an example of JTF uses, a partial test code of the seller agent from the book trading system is presented in Figure III-10 and Figure III-12. In the first test case (Figure III-10), is a unit test of the seller's start-up inner actions (**setup** method, Figure III-11). It begins by creating a mock of the DFService and a spied instance of the seller agent, initialized with three book titles and their corresponding prices (lines 49-53, Figure III-10). The **addBehaviour()** method of the spy is stubbed (line 54, Figure III-10), to ensure that no behaviours are added during the test allowing the focus to remain on the agent's startup process. After launching the agent, the test verifies that the **addBehaviour()** method is called twice (as in lines 56 and 57, Figure III-11), capturing the arguments to assert the correct behaviour types (lines 84-86, Figure III-10).

Subsequently, the test checks the seller agent's registration with the DFService (lines 44-55, Figure III-11), confirming that it solicits the mocked service appropriately and publishes the correct service description (lines 63-74 Figure III-10). Finally, the test validates the agent's catalogue to ensure it initializes correctly with only two books, accounting for an erroneous price that prevents the third book from being added (Lines 76-81).

```
42 @AUT(BookSellerAgent.class)
43 @PrepareForTest({ DFService.class, BookSellerAgent.class })
44 public class BookSellerAgentTest extends UJadeTest {
45   // DFService
46   @Role("Setup")
47   @Test
48   public void setuprTest() throws Exception {
49   PowerMockito.mockStatic(DFService.class);
50   String[] args = new String[] { "The Hobit", "10.25","The Alchemist", "12.45",
51                                  "History Of time", "xxx" };
52   BookSellerAgent seller = (BookSellerAgent) UJade.spyAgent(
53                                       BookSellerAgent.class, "Seller1", args);
54   doNothing().when(seller).addBehaviour(any(Behaviour.class));
55   ArgumentCaptor<Behaviour> behaviour = ArgumentCaptor.forClass(Behaviour.class);
56   ArgumentCaptor<DFAgentDescription> DFdesc =
57                                     ArgumentCaptor.forClass(DFAgentDescription.class);
58   UJade.startSpyAgent(seller);
59
60   verify(seller, timeout(5000).times(2)).addBehaviour(behaviour.capture());
61   // Published services verifications
62
63   PowerMockito.verifyStatic(DFService.class, timeout(5000).times(1));
64
65   DFService.register(eq(seller), DFdesc.capture());
66
67   assertEquals(1, DFdesc.getAllValues().size());
68   Iterator services = DFdesc.getValue().getAllServices();
69
70   ServiceDescription service = (ServiceDescription) services.next();
71   assertEquals("book-selling", service.getType());
72   assertEquals("JADE-book-trading", service.getName());
73   assertEquals(FIPANames.ContentLanguage.FIPA_SL, service.getAllLanguages().next());
74   assertFalse(services.hasNext());
75   // arguments verifications
76   Hashtable<String, Float> expectedCatalogue = new Hashtable<>();
77   expectedCatalogue.put("The Hobit", 10.25f);
78   expectedCatalogue.put("The Alchemist", 12.45f);
79
80   Hashtable catalogue = Whitebox.getInternalState(seller, "catalogue");
81   assertThat(expectedCatalogue, is(catalogue));
82
83   // add behavior verification
84   Behaviour[] beh = behaviour.getAllValues().toArray(new Behaviour[0]);
85   assertEquals("OfferRequestsServer", beh[0].getClass().getSimpleName());
86   assertEquals("PurchaseOrdersServer", beh[1].getClass().getSimpleName());
87
88   }
```

Figure III-10: Bookseller unit-level test case

Meanwhile, Figure III-12 presents a partial agent-level test case of the seller agent, with a test method (*testPurchaseOrderServer*) designed to test the purchase interactions between the seller and a buyer, from the seller side (Figure III-13). The testing scenario involves a mock buyer requesting two books: one that is available and one that is not. The seller is expected to respond with an *inform* message for the available book and a *failure* message for the unavailable one. Additionally, the seller's catalogue should be updated accordingly.

The test starts by creating and launching the seller and a mock buyer (lines 20-22, Figure III-12), then the test thread is suspended until the interaction between the seller and the mock buyer concludes (line 23, Figure III-12). The instruction at line 24, verify that the test was passed successfully from the mock's perspective. Finally, the updated state of the seller's catalogue is asserted (lines 26 and 27, Figure III-12).

On the mock buyer side, all interaction logic and verification instructions are implemented in a dedicated behaviour (the previous Figure III-6). The mock buyer initiates communication by submitting a purchase request for the book *"The Hobbit"* (lines 17-22, Figure III-6) and checks if the seller replies with an INFORM message (24-34, Figure III-6). It then issues a second request for the book "*Moonless*" (lines 36-38, Figure III-6) and verifies that the seller replies with a FAILURE message (41-47, Figure III-6). If all interactions and verifications are conducted correctly, the mock sets the test as passed (line 48, Figure III-6); otherwise, it marks the test as failed (line 52, Figure III-6).

```
22  public class BookSellerAgent extends Agent {
23
24     private Hashtable catalogue;
25     private String buyer;
26
27     @Override
28     protected void setup() {
29        catalogue = new Hashtable();
30        String targetBookTitle = null;
31        float targetBookPrice = 0f;
32        Object[] args = getArguments();
33        if (args != null && args.length >= 2) {
34             for (int i = 0; i < args.length; i = i + 2) {
35                try {
36                    targetBookTitle = (String) args[i];
37                    targetBookPrice = Float.parseFloat((String) args[i + 1]);
38                    catalogue.put(targetBookTitle, new Float(targetBookPrice));
39                } catch (NumberFormatException ex) {
40                    ex.printStackTrace();
41                }
42             }
43        }
44        DFAgentDescription dfd = new DFAgentDescription();
45        dfd.setName(getAID());
46        ServiceDescription sd = new ServiceDescription();
47        sd.setType("book-selling");
48        sd.setName("JADE-book-trading");
49        sd.addLanguages(FIPANames.ContentLanguage.FIPA_SL);
50        dfd.addServices(sd);
51        try {
52            DFService.register(this, dfd);
53        } catch (FIPAException fe) {
54            fe.printStackTrace();
55        }
56        addBehaviour(new OfferRequestsServer());
57        addBehaviour(new PurchaseOrdersServer());
58     }
```

Figure III-11: Bookseller agent

```
14  @AUT(BookSellerAgent.class)
15  public class BookSellerBehaviourTest extends Jat4 {
16
17   @Role("PurchaseOrdersServer")
18   @Test
19   public void testPurchaseOrdersServer() {
20   registerAndStartAgent("Seller2", BookSellerAgent.class,
21                new Object[] { "The Hobit", "10.25", "The Alchemist", "12.45" });
22   registerAndStartMockAgent("Buyer_Mock1", BuyerMock_ST.class, new Object[] { "3" });
23   AgentMonitorServices.waitUntilTestHasFinished("Buyer_Mock1");
24   assertMockAgent("Buyer_Mock1");
25
26   Hashtable bookcatalogue = (Hashtable) getAgentBelief("Seller2", "catalogue");
27   assertFalse(bookcatalogue.containsKey("The Hobit"));
28   }
```

Figure III-12: The bookseller agent-level test case

```
114 private class PurchaseOrdersServer extends CyclicBehaviour {
115
116  @Override
117  public void action() {
118      MessageTemplate mt = MessageTemplate.MatchPerformative(
119                                                  ACLMessage.ACCEPT_PROPOSAL);
120      ACLMessage msg = myAgent.receive(mt);
121      if (msg != null) {
122
123          String title = msg.getContent();
124          ACLMessage reply = msg.createReply();
125
126          Float price = (Float) catalogue.remove(title);
127          if (price != null) {
128                 reply.setPerformative(ACLMessage.INFORM);
129                 myAgent.send(reply);
130                 buyer = msg.getSender().getLocalName();
131          } else {
132              reply.setPerformative(ACLMessage.FAILURE);
133              reply.setContent("not-available");
134              myAgent.send(reply);
135          }
136      } else
137      block();
138
139  }
140 }
```

Figure III-13: The bookseller's behaviour: purchase-order server

5 | Assessment of JTF

To showcase the effectiveness of JTF in testing JADE agents, an experimental comparison was carried out among the testing capabilities of JTF, JAT alone, and UJade alone. The following questions were investigated:

Q1.1: **Does the integration of UJade with JAT in JTF provide superior agent testing compared to UJade alone??**

Q1.2: **Does the integration of UJade with JAT in JTF result in better agent testing compared to JAT alone?**

In the experiment, the 23 agents, from the pre-retained sample of MAS (Chapter I: 6 |), were tested using each solution. Then, the efficacy of each solution's test suite was assessed using the mutation testing technique (MTT) (DeMillo et al., 1978; Sayward et al., 1978). The MTT is recognised as a reliable indication of test quality. In contrast to coverage metrics techniques, which

merely indicate if a line or instruction has been tested, MTT assesses the extent to which that instruction is effectively tested (Chekam et al., 2017; Inozemtseva & Holmes, 2014).

During the planning phase of the experiment, it became evident that certain sections of agent code could tested (with 100% code coverage) at either the unit level or agent level (as detailed in the test design technique section 5.2.1 |). These sections include:

During the experiment planning, it was found that some pieces of agent's code can be completely tested (100 per cent code coverage) either at the unit-level or agent-level (see test design technique section):

- Agent behaviours implementing a one-way communication schema (either sending or receiving messages).
- Agents implementing predefined JADE interaction protocols.

These dual-test-level codes are primarily communication-oriented, suggesting they should be tested at the agent level. However, the capabilities of UJade also facilitate testing at the unit level. This led to the formulation of an additional question:

Q2**: Which testing level—unit-level with UJade or agent-level with JAT—requires less effort to write tests for the dual-test-level cases?**

5.1 | Experimental procedure

Due to the dependency between the experiment's questions, they were investigated in reverse order, starting with Q2, then the main questions, Q1.1 and Q1.2.

5.1.1 | Answering Q2

To address this question, the 23 agents have been reviewed to identify potential dual-test levels. For the identified cases, tests were implemented at both the unit level with UJade and the agent level with JAT, following the strategy presented in the next section (5.2 | .Then, the effort to construct these tests was assessed and analysed to determine the most efficient approach for testing these cases.

To quantify the effort involved in writing tests (Test Construction Effort), the definition proposed by Bruntink & Deursen (2006) was utilised. This definition considers the size of the test suite as a significant indicator of testing effort. Bruntink and Deursen suggested two primary metrics to measure test size: the number of test lines of code (TLOC) and the number of JUnit assert methods (NbAssert) employed in testing a class.

Recognising the unique aspects of the Agent-Oriented (AO) paradigm, two supplementary metrics were adopted: the number of test cases (NTC) and the number of mocks utilised in testing an

agent (NbMock). This latter metric accounts for the total number of mocks and spies created using JAT and PowerMockito, respectively (equation Eq.III-1).

Furthermore, the definition of the NbAssert metric was slightly adjusted to reflect JTF's capabilities. JTF not only allows assertions on the agent's internal state but also facilitates verification of the agent's internal logic. Consequently, the count of Mockito verify instructions invoked within an agent's test is also included in the NbAssert metric (equation Eq.III-2). This nuanced approach ensures a comprehensive evaluation of testing effort, providing valuable insights into the testing process within the context of the AO paradigm.

$$\text{NbMock} = \text{Nb\_JAT\_Mock} + \text{Nb\_Spy} + \text{Nb\_Mock} \quad \textbf{Eq.III-1}$$

$$\text{NbAssert} = \text{Nb\_Assert} + \text{Nb\_Verify} \quad \textbf{Eq.III-2}$$

Ultimately, to facilitate testing-effort comparison an overall testing-effort metric is formulated as a weighted sum function of the previous four metrics (formula Eq.III-3). The function's weights were established via the rating method: Analytic Hierarchy Process (AHP) (Saaty, 1990). The latter was selected due to its low computation time, and the ease of checking the consistency of the adopted weights.

All AHP-related calculations were performed using **PriEsT**[22] tool **(**Priority Estimation Tool) (Siraj et al., 2015). Table III-1 shows the pairwise comparison matrix. Although, the AHP approach suggests a nine-point fundamental scale for comparisons (Table II-5), a 5-point scale (1 to 5) was used, as the higher levels (7 to 9) are intended for comparison decisions based on demonstrated claims and facts, which was not the case here. The consistency rate of the obtained wights is 0.009, which is below the recommended threshold of 0.1 (Saaty, 1990).

$$Test\_Effort = 0.449 \times NbMock + 0.235 \times TLOC + 0.235 \times NTC + 0.081 \times NbAssert \quad \textbf{Eq.III-3}$$

	NTC	TLOC	NbMock	NbAssert
NTC	-	0.5	1	3
TLOC	2	-	2	5
NbMock	1	0.5	-	3
NbAssert	0.33	0.2	0.33	-

Table III-1: AHP pairwise comparison matrix

As a metrics measurement tool, the JMP tool, presented in Chapter II: 5 | was upgraded to handle the JTF test code, and to measure automatically the previous four test metrics. Like the weighed sum functions of The JADE measurement model (Chapter II: 6 |), the testing effort equation was not hard coded within the tool, the tool generates a second CSV file with the test metrics

[22] https://sourceforge.net/projects/priority/

values of the agents under analysis. For this question analysis, the CSV file was processed via a Python script.

5.1.2 | Answering Q1.1 and Q1.2

To address this experiment's main questions, the 23 agents were tested following section 5.2 | 's strategy, at both levels: unit and agent, using UJade and JAT respectively. The resulting test cases were then categorised into three distinct groups:

- **UJade Category**: This contained the unit-level test cases implemented using UJade.
- **JAT Category**: This included the agent-level test cases implemented using JAT.
- **JTF Category**: This encompassed test cases from both levels. To avoid duplication arising from the dual-test-level codes, the findings from the earlier analysis (Q2) should employed to identify the less effort-intensive testing level and, consequently, which test cases to retain.

The test effectiveness of each category was assessed via the mutation testing technique (MTT). This technique entails creating defective iterations of the agents under test, known as mutants, by introducing small syntactic alterations (mutations) in the code. The test suite for each category was then executed against these mutants to determine the category's tests' ability to detect faults. If a test fails on a mutant, the mutant is deemed as "killed"; if not, is considered as "live." The proportion of killed mutants to the overall number of generated mutants is the mutation score (DeMillo et al., 1978; Sayward et al., 1978). A higher mutation score signifies a more effective test suite with strong testing capabilities and high fault detection (Chekam et al., 2017; Just et al., 2014; Frankl et al., 1997).The mutation testing was executed using the PIT tool[23] (Henry Coles et al., 2016) , that auto-generating mutants and assessed test suits capabilities to identify them.

Since the mutation test results quality heavily depends on the strength of the mutation process (Chekam et al., 2017), a list of 15 mutation operators (mutators) was retained. Table III 10 lists these retained mutators[24]; a more detailed description can be found in (Henry Coles, 2020). The first 13 mutators are categorised as "**stronger group**" configuration according to PIT documentation (Henry Coles, 2020). The two additional one (Non-Void Method Calls and Inline Constant) were incorporated to better align the test with the characteristics of the JADE agent code.

The "*Inline_Constant*" mutstor helps to check if a tests case can detect changes in the agent's initial state, since in JADE agents, beliefs are sometimes defined as constant variables with inline

[23] http://PIT.org/, version 1.6.2, https://mvnrepository.com/artifact/org.PIT/PIT-maven/1.6.2
[24] PIT was configured with mutators options: STRONGER, NON_VOID_METHOD_CALLS, INLINE_CONSTS

initialization instructions. Where "*Void_Method_Calls*" with "*Non_Void_Method_Calls*" help to mutate agents' internal actions, for example,

- **Mutating agent behaviour**: by removing calls to methods that add or remove behaviours (e.g., addBehaviour(), addSubBehaviour()), and eliminating calls to methods that manipulate the agent's life state (e.g., doDelete(), block(), doSuspend(), doWait()).
- **Mutating of agent communication**: by blocking message sending (removing calls to the send()), altering received messages (changing the returned value of Receive() and blockReceive()), and mutating messages sent or received (altering receptors, message content, and performative, etc.).
- **Mutation of agent interaction with the platform**: by modifying communication with DFService (removing calls to register(), altering the description of services to advertise, changing the result of DF searches), and blocking the creation and launching of new agents.

Mutator	Description
Conditionals Boundary	Replace the relational operators (<, <=, >, >=): "<" to "<=", "<=" to "<", ">" to ">=",">=" to">"
Increments	Replace increments instruction with decrements and vice versa.
Invert Negatives	Inverts the negation sign of integer and floating-point numbers
Math	Replace binary arithmetic operations for either integer or floating-point arithmetic with another operation:"+" to "-", "*" to "/", "% " to "*","&" to "|","<<" to ">>", ">>>" to "<<"
Negate Conditionals	Mutate the Boolean conditionals: "= =" to "! =", "!=" to "= =" , "<= " to ">",">= " to "<", "< " to ">=", ">" to "<="
Empty returns	Replace return values with an 'empty' value for types : String->"" , Optional -> Optional.empty(), List -> Collections.emptyList(), Set -> Collections.emptySet() ,Integer -> 0, Short -> 0, Long -> 0,Character -> 0, Float -> 0, Double -> 0
False Returns	Replace primitive and boxed Boolean return values with false.
True returns	Replace primitive and boxed Boolean return values with true.
Null returns	Replace return values with null
Primitive returns	Replace int, short, long, char, float and double return values with 0.
Remove Conditionals	Remove all conditional statements such that the guarded statements or if If an else block is present it will always execute
Experimental Switch	Mutate the switch statement by replacing first the default label (if it is used) with the first label in this switch statement then All the other labels with the old default one.
Void Method Calls	Remove method calls to void methods
Non Void Method Calls	Remove method calls to non-void methods. Their return value is replaced by the Java Default Value for that specific type: Boolean with false, int, byte, short and long with 0, float and double with0.0, a chart with '\u0000' and Object with null.
Inline Constant	mutates the inline constant instruction by replacing the literal value assigned this constant, for instance, Boolean constant: replace the unmutated value true with false and replace the unmutated value false with true.

Table III-2:Mutation operations

5.2 | Testing strategies

A comprehensive unified testing strategy was implemented to ensure a robust foundation for this empirical analysis and mitigate potential internal validity threats. This strategy includes the following components:

- **Common Test Design Technique:** A standardised test design technique based on McCabe's structured test approach (McCabe, 1976b) was adopted to guide the development of test cases.
- **Test Completion Criterion:** The criterion for test completion was set to achieve 100% McCabe (complexity) coverage, ensuring that all basis paths are thoroughly tested. This high standard aims to minimise undetected faults and maximise reliability.
- **Coverage Measurement Tool:** The EclEmma[25] tool was selected for its capability to accurately compute the code coverage of the agents.

5.2.1 | Test design technique

the McCabe technique utilises the program's control flow structure to determine path coverage criteria (Watson et al., 1996), ensuring that each program statement is covered at least once (Bharat Kumar et al., 2012; Watson et al., 1996) and requires fewer test cases than branch testing technique.To maintain uniformity in test-case designing among all agents of the systems under test (SUT), a comprehensive set of guidelines is established for each testing level. At the unit level, the following AUT elements of agents' methods, behaviours, sub-behaviours, and methods of behaviours, are tested following the procedure in Figure III-14.

```
For each AUT
  Agent's Methods //except agent Setup method
    Verify inner-actions     //other methods calls
    Assert fields, beliefs and method return value.
    If the method contains communication code    // send & receive actions
      Verify only the number of sent messages and messages' performatives.
      Verify the effect of message reception, by faking the message reception.
    Test exceptions    // if excite
    Test exception catch block // if it isn't empty and it influences the under paths
  Agent's behaviour & Setup method
    Assert passed arguments
    Verify added behaviours  & sub-behaviours   // type and arguments
    Verify Interaction with the DFService
      Verify service requesting and advertising by mocking or spying on DFService
    Verify inner methods' calls
    Test exceptions (if they excite)
    Test exception catch block // if it isn't empty and it influences the under paths
```

[25] https://www.eclemma.org/

```
18      If its interaction-oriented behaviour  // i.e. mainly communication code
19        If it is a one-way communication /// Dual-test-level
            // Agent only sends or receive messages, but not both
            // This case can be tested on both levels unit and agent.
20          If a decision is made to test it at unit level:
21            Test the possible interaction scenario.
22            Verify the effect of message reception, by faking the msg reception.
23            Verify the messages sent in terms of number and order.
24            Verify sent message's conformance   //performative, content, etc.
25        Else, it is multiple ways interaction
26          Verify only the start condition   // fake message reception if needed
27          Verify inner action   // those that are not related to communication
28          Assert beliefs changes // if not related to the behaviour end condition
29      Test all behaviour's methods   // follow Agent's methods test steps
30      Test all behaviour's sub-behaviours // follow Agent's behaviours test steps
```

Figure III-14 Agent's units test steps

At the agent level, given that behaviours are the primary building blocks of agents in JADE (Bellifemine et al., 2007), each AUT behaviour with a communication code is tested by identifying all possible interaction scenarios using the McCall technique. These scenarios are then tested using JAT mock agents. Each scenario-under-test (ScUT) is tested following the procedure outlined in Figure III-15.

```
1  For each ScUT
2     In mock agent
3         Assert the conformance of interaction
              // received messages sequence, number, and order
4         Assert the correctness of received messages.
              // performative, content, language, etc.
5     In test method
6         Assert that the agent mocks had terminated with success
7         Verify the interaction effects on AUT's beliefs
```

Figure III-15: Agent level test steps

The dual-test-level situations were identified during the guideline's definition. The first one was identified within behaviours that are mainly communication-oriented and their communication schema is simple: only one-way communication, either receive or send messages. This situation is intended to be tested at agent level, flowing Figure III-14 guidelines, but can also be tested at the unit level by faking messages receptions and then testing their effects on agent inner state (in case of message reception), or by verified agent send method calls' arguments (in case of message sending).

The second dual-test-level situation pertains to testing JADE's agent interaction protocols. JADE API[26] offers pre-made interaction protocol classes (defined as agent behaviours) to simplify protocols implementation and ensure message flow consistency. Thies protocole can be customised, based on the interaction context, by overwriting methods for preparing the messages to send (*prepareRequestes(), prepareResultNotificatione(), prepareCFP()*, etc.); or processing received ones

[26] https://jade.tilab.com/doc/api/index.html

(*handleAgree(), handleInfome(), handleAllResponses()*, etc.), These protocols can be tested at the agent level using JAT mocks, like other interaction behaviours, or at the unit level, by just focusing on the overwritten methods.

6 | Results of Assessment

6.1 | Dual-test-level cases

The review of the code of the 23 agents under study had revealed 30 dual-test-level cases, Table III-3 and Table III-4 indicate their location in the SUTs. Overall, 163 test cases were developed to investigate them and to answer Q2, comprising 126 at unit-level testing with UJade and 37 at agent-level testing with JAT. The distribution of these test cases is presented in Table III-5.

MAS	AUT	Case
3	BidderComp	C 1
	BidderHuman	C 2
5	HospitalAgent	C 3

Table III-3: One-way communication cases

MAS	AUT	Case
1	CarrierAgent	C1-C2
	CompanyAgent	C3
5	PatientAgent	C4
6	Building	C5
	Bulb	C6-C9
	Controller	C10
	LightSensor	C11-C15
	MeshNetGateway	C16
	Room	C17-C18
	TempSensor	C19-C23
	ToggleSwitch	C24-C27

Table III-4: interaction protocol cases

Dual-test-level case-type	N:	Unit level	Agent level
One-way communication	3	3	3
Interaction protocols	27	123	34
Sum	**30**	**126**	**37**

Table III-5: Test-cases per dual-test level case-type

The collected data indicates that one-way communication cases required fewer test cases per dual-test-level compared to interaction protocol cases. This is due to the simplicity of the communication schema in one-way communication (either sending or receiving messages), which can be easily tested at the unit level by mocking message reception or spying on the agent to verify message sending; or at the agent level by implementing a simple reactive JAT mock agent.

Interaction protocol cases, however, required more test cases, approximately six per case. Additionally, the number of test cases developed at the unit level was three times higher than at the agent level. This is because, with UJade at the unit level, each method within the interaction protocol was tested separately, with an average of three methods per protocol. In contrast, with JAT at the

agent level, the interaction protocol was tested as a single entity via JAT mocks, allowing each test case to verify multiple interaction scenarios simultaneously.

6.1.1 | One-way communication

Figure III 23 displays the testing-effort metrics for the three one-way communication cases at each level. The overall effort values, derived from formula Eq.III-3, are shown in a separate green bar chart.

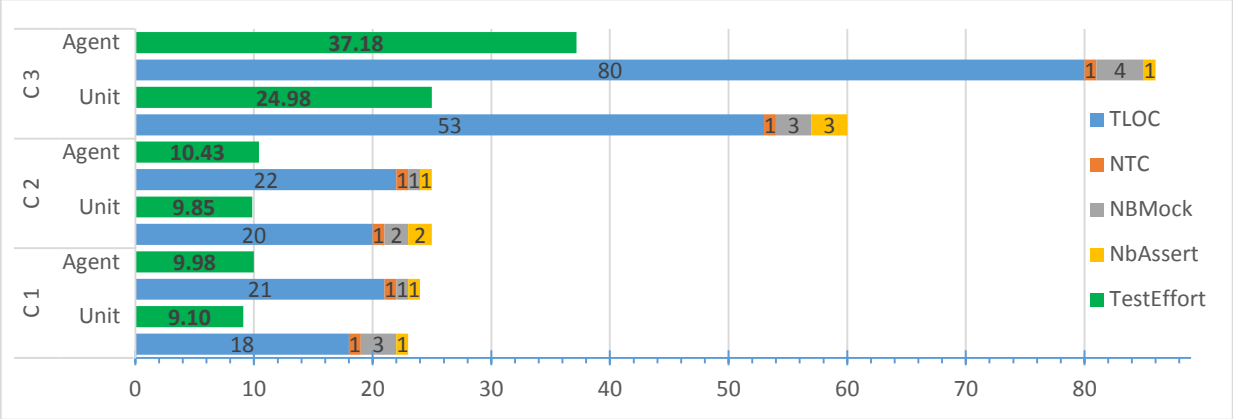


Figure III-16: One-way communication testing effort's metrics (unit vs agent level)

Testing one-way communication cases at the unit level with UJade required less effort than at the agent level with JAT. The TLOC (Total Lines of Code) for UJade is lower than for JAT. This is because JAT's mocks are hand-coded JADE agents, while UJade's mocks and spies are automatically generated using Powermockito, requiring less code to define their behaviours. Additionally, the number of assertions at the unit level is generally higher than at the agent level. Unlike JAT, which only verifies message conformance and the effects on the agent's beliefs, UJade at the unit level allows for more comprehensive verification, including the agent's inner actions.

Table III-6 shows the average values of the testing effort measurements for the three one-way communication cases. The data indicates that testing this schema's code at the unit level with UJade reduces the number of test lines of code by 26% and allows for better verification capabilities due to the increased number of assertions and verification instructions. However, this approach requires 33% more mocks compared to the agent level. Overall, the testing effort is reduced by 24% when using UJade compared to JAT. Due to the small sample size (only 3 cases), these findings couldn't be statistically verified.

Test Framework	TLOC	NbAssert	NbMock	NTC	Testing effort
UJade	30.33	2.00	2.67	1.00	14.65
JAT	41.00	1.00	2.00	1.00	19.20

Table III-6: The average testing efforts measurements values –One-way communication-

6.1.2 | Interaction protocol

Figure III-17 and Figure III-18 present the testing effort's metrics and the overall effort values of the 27 interaction protocol cases across both levels. Similar to one-way communication cases, the testing efforts for interaction protocol cases show lower TLOC values and higher NbAssert values at the unit level. As previously mentioned, JAT requires more code to implement mocks and can only formulate assertions on AUT's messages and beliefs.

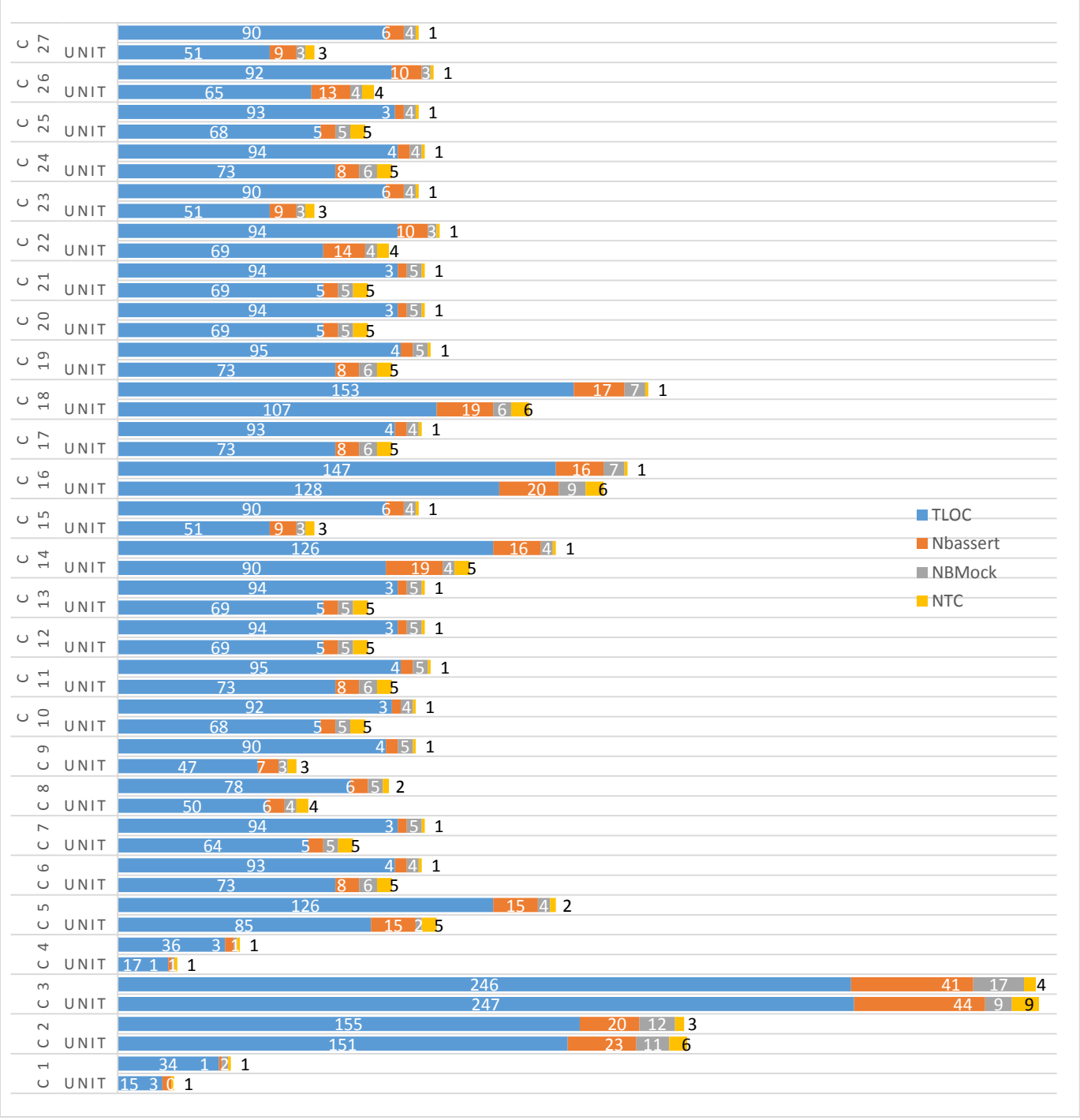


Figure III-17: Interaction protocols cases Testing effort's metrics (unit vs agent level)

However, unlike the previous dual test-level cases, interaction protocols required more test cases at the unit level compared to the agent level, with the number of mocks being nearly the same.

The exception is observed in case 3 (C3), where the complexity of the interaction scenarios (high TLOC and NAssert) necessitated 17 mocks in UJade compared to 9 in JAT. This case concerns am iterative contract net protocol, used by the company agent (AUT) to interact with multiple bidders who underbid each other in hopes of others withdrawing. Testing it required a high number of JAT agent mocks (17) to cover different interaction scenarios. At the unit level, fewer mocks (only 9) were needed, as UJade offers functionalities to directly call and test the interaction protocol methods separately.

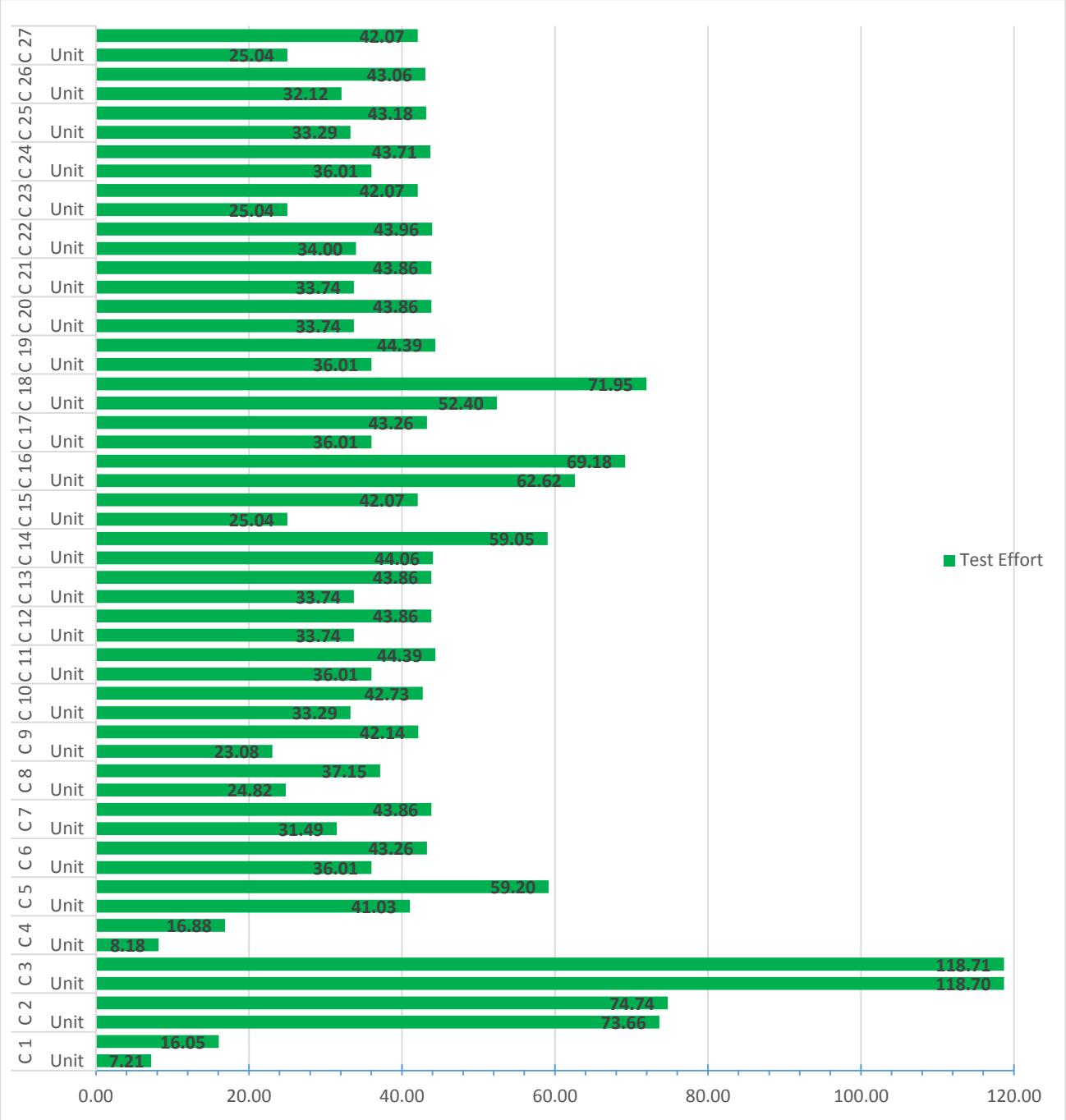


Figure III-18: Interaction protocols cases the overall Testing effort values (unit vs agent level)

Table III-7 shows the average values of the overall testing-effort measurements. Testing interaction protocols at the unit level proves to be more efficient than at the agent level, requiring

25% less test code and offering better verification capabilities, with nearly the same number of mocks. However, the number of test cases needed is 2.5 times higher than with JAT. Overall, this reduces the testing effort by 22%.

Test Framework	NTC	TLOC	Assert	Mocks	Testing effort
UJade	4.56	76.48	10.59	4.85	37.41
JAT	1.26	102.67	8.07	5.07	48.24

Table III-7: Average testing effort metrics value – interaction protocols-

To assess statistically the significance of these findings, a lower-tailed Mann-Whitney test was applied to the overall testing effort values with the following hypotheses (alpha = 0.5):

- H0: Testing interaction protocols at the unit level is not less effort consuming than at the agent level.
- H1: Testing interaction protocols at the unit level is less effort consuming than at the agent level.

The results of the statistic test execution on the XLSTAT tool[27] are shown in Table III-8, indicating a p-value of < 0.0001, which is lower than the significance level alpha of 0.5. Consequently, the null hypothesis H0 is rejected, and the alternative hypothesis H1 is accepted. This confirms that testing interaction protocols at the unit level is less effort consuming, and the difference is statistically significant.

U	155
Expected value	364.500
Variance (U)	3333.481
p-value (one-tailed)	< 0.0001
alpha	0.05

Table III-8: Mann-Whitney test (U) results

From above analysis, it is evident that testing one-way communication schemas and interaction protocols using UJade demands less effort compared to using JAT.

6.2 | **Testing capabilities comparison**

A total of 431 test cases were developed —360 at the unit level and 71 at the agent level—, categorised as follow: **UJade Category** with the 360 unit-level test cases, **JAT Category** with the 71 test cases implementable under JAT and JTF Category with 394 test cases (360 at the unit level and 34 at the agent level) with the dual test-level code being tested at the unit level, following the previous findings.

Table III-9 displays the code coverage (instruction and complexity) for each category's test suite. When interpreting the collected coverage data, it is important to consider the following facts:

[27] https://www.xlstat.com

- EclEmma uses a modified version of cyclomatic metrics (strict cyclomatic metric), different from the ordinary cyclomatic metric used for test case design. In the strict cyclomatic metric, the side effects of Boolean operators in binary decisions—such as "if" and "while" statements—are considered, with each Boolean operator incrementing the complexity by one.
- EclEmma does not account for exception handling as branches (try/catch blocks) when calculating complexity[28].
- In the test case design guidelines, it was decided not to test empty catch blocks or catch blocks with only one instruction (i.e., displaying the exception trace in the console). This decision assumed that these blocks are not worth testing since they do not involve any inner-agent action or belief manipulation.

The first fact explains why the complexity coverage is not 100% for some agents, even though the test cases were designed to meet this criterion. For instance, the BookBuyerAgent has 100% instruction coverage in the JTF suite but only 92% complexity coverage. The latter two facts clarify why, in cases marked with (*), the complexity coverage is 100% while instruction coverage falls short.

MAS	Agent	Nbr Instr	Instruction Coverage			Complexity	Complexity Coverage			Tag
			JAT	UJade	JTF		JAT	UJade	JTF	
1	CarrierAgent	124	85%	98%	98%	7	57%	86%	86%	
1	CompanyAgent	685	91%	99%	99%	22	86%	100%	100%	*
2	AuctionAgent	401	100%	61%	100%	12	100%	58%	100%	
2	BargainerAgent	462	97%	99%	99%	17	82%	100%	100%	*
2	EnterpriseAgent	436	99%	89%	99%	18	94%	83%	94%	
3	Auctioneer	746	96%	93%	100%	44	75%	86%	93%	
3	BidderComp	330	96%	55%	99%	14	93%	79%	100%	*
3	BidderHuman	398	96%	99%	99%	21	95%	100%	100%	*
4	BookBuyerAgent	522	92%	88%	100%	38	74%	76%	92%	-
4	BookSellerAgent	242	96%	60%	99%	14	86%	64%	93%	
5	HospitalAgent	612	99%	61%	99%	35	80%	54%	83%	
5	PatientAgent	1118	96%	70%	98%	65	74%	74%	91%	
6	Building	268	77%	95%	96%	14	71%	100%	100%	*
6	Bulb	957	87%	96%	96%	42	74%	95%	95%	
6	Controller	216	97%	97%	97%	9	89%	100%	100%	*
6	LightSensor	909	96%	97%	97%	35	74%	89%	91%	
6	MeshNetGateway	368	86%	96%	96%	17	94%	100%	100%	*
6	Room	494	93%	97%	97%	22	77%	100%	100%	*
6	TempSensor	930	97%	97%	97%	31	81%	97%	97%	
6	ToggleSwitch	685	97%	97%	97%	28	79%	96%	96%	
6	Demo	420	0%	94%	94%	8	0%	100%	100%	*
7	Player	362	89%	100%	100%	28	82%	100%	100%	
7	GameMaster	260	94%	100%	100%	18	89%	100%	100%	

Table III-9: Agent's Code Coverage JAT vs UJade vs JTF

The data indicates that combining both UJade and JAT (i.e., JTF) achieves better code coverage than using JAT or UJade alone. In the worst case, JTF matches the code coverage of JAT (in 2 agents) or UJade (in 14 agents). But since the code coverage metrics can be misleading indicators of test quality. The mutation test was performed instead.

[28] https://www.eclemma.org/jacoco/trunk/doc/counters.html

Based on the retained mutation list, PIT introduced 3794 mutations in the 23 agents' code, averaging nearly one mutation per LOC (0.89 per LOC). Due to flaky tests, the mutation test was run ten times. Each run produced slightly different mutation scores. However, the computed standard deviations of the overall mutation scores (covering all seven systems per run) were small—0.64% for UJade, 0.66% for JAT, and 0.80% for JTF. This minimal variability could be neglected without impacting the analysis's findings. These flaky tests were primarily caused by concurrent execution between agents' (AUT and mocks) threads and the JTF test runner thread, as well as the heavy use of Synchro Wait[29] routines in both the JADE platform and JTF. These issues are well-documented sources of flakiness in multithreaded systems (Lam et al., 2020; Eck et al., 2019; Luo et al., 2014).

Table III 11 displays the mutation score for each category per agent, along the number of mutations. The JTF test suite achieved a better mutation score or, in the worst cases, matched the scores of JAT-alone or UJade-alone test suites. In 7 cases (tagged as T), the JTF test suites outperformed both JAT and UJade test suites. In 12 cases (tagged as U), the JTF test suites matched the scores of UJade, while in one case (tagged as J), it matched the score of JAT. In three cases, all suites scored the same. When comparing JAT and UJade mutation scores, JAT had better scores in 7 cases, whereas UJade outperformed JAT in 13 cases.

Moreover, the high mutation scores validate the quality and rigor of the adopted test design technique. Despite the robust mutation test conditions (0.89 mutations per LOC and 15 classes of mutation operations), all mutation scores exceeded 95%. The median mutation scores for the implemented test suites were 96% for UJade, 96% for JAT, and 99% for JTF.

MAS	Agent	Nb Mutations	Mutation Score			Tag
			UJade	JAT	JTF	
1	CarrierAgent	122	99%	93%	99%	U
1	CompanyAgent	176	99%	93%	99%	U
2	AuctionAgent	125	56%	100%	100%	J
2	BargainerAgent	144	99%	97%	99%	U
2	EnterpriseAgent	153	87%	99%	99%	T
3	Auctioneer	159	92%	97%	99%	T
3	BidderComp	67	60%	96%	99%	T
3	BidderHuman	72	99%	93%	99%	U
4	BookBuyerAgent	166	89%	87%	100%	T
4	BookSellerAgent	80	35%	96%	99%	T
5	HospitalAgent	206	50%	99%	100%	T
5	PatientAgent	384	69%	98%	98%	T
6	Building	81	96%	81%	96%	U
6	Bulb	306	96%	87%	96%	U
6	Controller	84	98%	98%	98%	-
6	LightSensor	344	97%	97%	97%	U
6	MeshNetGateway	87	94%	89%	94%	U
6	Room	171	97%	95%	97%	U
6	TempSensor	351	97%	97%	97%	-
6	ToggleSwitch	240	98%	98%	98%	-
6	Demo	118	93%	0%	93%	U

[29] make asynchronous calls without properly waiting for the call to return

MAS	Agent	Nb Mutations	Mutation Score			Tag
			UJade	JAT	JTF	
7	Player	92	100%	86%	100%	U
7	GameMaster	66	100%	91%	100%	U
	Sum	**3794**				

Table III-10: Mutation Test Results

To validate these results and assess the statistical significance of JTF's performance over JAT and UJade, two upper-tailed Wilcoxon signed-rank tests (alpha=0.5) based on the following hypotheses were executed using XLSTAT tool

Test 1:

- H0.1: JTF does not allow a better mutation score than UJade.
- H1.1: JTF has a better mutation score than UJade.

Test 2

- H0.2: JTF does not allow a better mutation score than JAT.
- H1.2: JTF has a better mutation score than JAT.

Table III-11 and Table III-12 present the results of the tests. The p-values are lower than the alpha significance level (0.005 and less than 0.001, respectively). Consequently, the null hypothesises H0.1 and H0.2 are rejected, while the alternative hypotheses H1.1 and H1.2 are accepted.

Based on the analysis, it can be concluded, as response to the main research questions (Q1.1 and Q1.2), that the integration of UJade with JAT (i.e., JTF) results in more effective testing farmwork compared to using either JAT or UJade alone.

V	36
Expected value	18.000
Variance (V)	51.000
p-value (one-tailed)	0.005
alpha	0.05

Table III-11: Wilcoxon signed-rank test (V) 1 results

V	190
Expected value	95.000
Variance (V)	617.500
p-value (one-tailed)	< 0.0001
alpha	0.05

Table III-12:Wilcoxon signed-rank test (V) 2 results.

7 | Threat to Validity

This chapter is the second milestone in the main testability investigation process. The aim was to build a complete and effective testing framework for JADE's agent and use it to develop test cases for systems under study. To minimise the JTF formwork construction threats, it was built upon

well-known testing solutions for Jave-based systems: Junit, Mockito, and PowerMockito, and integrated an existing solution for the JADE agent interaction testing, JAT.

Furthermore, the formwork effectiveness was demonstrated through an empirical experiment that was also meticulously planned and executed to minimise potential threats to its validity. For instance, the mutation-test technique was used instead of coverage metrics to evaluate JTF effectiveness, as it is a good indicator of test effectiveness. Standard metrics for code coverage, testing effort, and mutation test scores were adopted, along with a set of automated tools to collect measurements (EclEmma, and JMP), execute statistical tests (XLSTAT), and execute all steps of the mutation test —mutation generation, test execution— (PIT tool).

Furthermore, the strengths of conclusions about JTF's effectiveness compared to JAT or UJade alone, and the efficiency of UJade compared to JAT in interaction protocol test writing were statistically demonstrated.

Regarding the second aim of this chapter, which is implementing tests for the retained agents, they were meticulously written as part of the JTF efficacy experiment, with a rigorous strategy in place to minimize potential threats to their construction. The McCabe technique was employed, to guarantee thoroughness and uniformity of tests, ensuring that each instruction is tested at least once. Guidelines were also established to assist the test developer, a D.Sc candidate with extensive experience in JADE agents and Java program testing. Furthermore, all tests code was reviewed after three months by the same developer to identify any errors. The high mutation test scores obtained in the section 6.2 | (all above 95%), despite the robust mutators list (0.89 mutations per LOC), attest to the quality and rigour of the test implementation process.

8 | Conclusions

This chapter tackles the second critical issue in the testability investigation, thus the lack of a comprehensive testing framework for JADE agents at both unit and agent levels. The literature and the open-code repository review revealed that existing solutions primarily target agent interaction debugging and testing, but unit testing is largely overlooked. The proposed solution within this chapter is a framework constituted of two components: the JAT4 component, and the UJade component. JAT4 is an enhanced version of the well-known JAT framework for JADE’s agent interaction testing framework. Whereas the UJade component is a new solution that was built upon JUnit and PowerMockito with aims to effectively test JADE’s agent. Additionally, JTF introduces a DSL for agent test writing, designed to facilitate test development and to improve the quality of test code’s units. Additionally, JTF introduces a DSL for agent test writing, designed to facilitate test development and improve the quality of test code.

The usability and effectiveness of JTF were thoroughly evaluated in real-world scenarios. An empirical study involving two experiments was conducted on a retained sample of agents. In the first experiment, the study focused on agents' codes that could be tested at both unit and agent levels. The results demonstrated that, in these special cases, UJade required less effort for test writing compared to JAT. The second experiment assessed the quality of tests developed using JTF through the mutation-test technique. The findings concluded that JTF allows for the creation of tests with higher effectiveness and efficiency compared to using UJade or JAT alone.

Chapter IV: Testability Investigation

1 | Introduction

After addressing the primary issues hindering the agent testability investigation in the previous two chapters (Chapter II: Chapter III:), this chapter delves into the core of the thesis. It investigates JADE agent testability by seeking answers to the main research questions:

- What agent properties influence its testability, and on which testing level (unit or agent) does this occur?
- What agent attributes (capabilities) influence its testability, and on which testing level (unit or agent) does this occur?
- Can agent testing effort be predicted from its source code?

This chapter is organized as follows: Section 2 details the research methodology, dependent and independent variables, and the procedures for data collection and analysis. Section 3 presents and discusses the study's results. Section 4 addresses the threats to validity. Section 5 offers a summary of the conclusions and proposes directions for future research.

2 | Research Methodology

In the quest for answers to the main research questions, the latter had to be refined in more atomic ones than can be empirically answered from agent code (both source and test code). The refined ones are:

*Q 1. Is there a relationship between **agent-oriented metrics** and **testing effort metrics**?*
*Q 2. Is there a relationship between **agent-oriented metrics** and **testing level**?*
*Q 3. Is there a relationship between **agent attributes** and **testing effort metrics**?*
*Q 4. Is there a relationship between **agent attributes** and **testing level**?*
*Q 5. Is there a relationship between **agent properties** and **testing effort metrics**?*
*Q 6. Is there a relationship between **agent properties** and **testing level**?*
*Q 7. What is the individual (and the combined) effect of **SC (Source Code) metrics** on agent **testing effort?***
*Q 8. What is the individual (and the combined) effect(s) of **agent attribute(s)** on agent **testing effort**?*
*Q 9. What is the individual (and the combined) effect(s) **of agent property(ies)** on agent **testing effort**?*
Q 10. *Could agent **testing effort** be predicted from **SC metrics***?

To answer the abovementioned questions, a research methodology of three stages was drawn. The first stage looks to answer the questions Q1 to Q6, the statistical correlation method is used to analyse the relationship between both the test metrics and testing levels, and each of the three

metrics sets: agent properties, attributes and source code (SC) metrics. Next, seeking answers to questions Q7 to Q9, the logistic regression methods are applied: the univariate logistic regression is used to evaluate the individual effect of each element of the sets: agent properties, attributes and SC metrics on agent testing effort; then the multivariate logistic regression is used to evaluate the combined effect of the metrics of each set on agent testing effort. In the last stage, aimed to answer question Q10, machine learning algorithms are used to develop prediction models of agent testing effort from agent-oriented metrics.

Whilst the definition of this research methodology, an essential question was raised. Agent properties (and subsequently attributes) are dynamic, an agent can balance their degrees as needed to deal with the variety of situations that may face (Winikoff, 2012; Alonso, Fuertes, Martinez, et al., 2010; Alonso et al., 2009), and studying agent testability without taking into account this fact can be seen as a threat to validity since the analysis can be less accurate. Handling this issue requires the use of dynamic metrics which is out of the scope of this study. The study aims to measure and predict agent testing effort from its source code where only static metrics are applicable.

To overcome this limitation a decision was made to undertake the study not at the agent abstraction level but at the agent's role level. An agent can play multiple roles and its properties' values when it is playing a specific role are more stable (less volatile, compared with the same properties values if agent behaviour as a whole is considered), since, at this level, the executed behaviour is more specific and goal-oriented.

JADE agent does not provide an explicit primitive to specify agent role, but generally, agent roles are implemented as behaviours (Bellifemine et al., 2007; Moraïtis et al., 2003; Massonet et al., 2002); thus, an assumption is made to consider each agent's top-level behaviour (behaviour and not sub-behaviour) as a role. Furthermore, the remaining agent code, that is, the code of agent's main class implementing like agent's fields, methods (including setup methods), etc. is considered as a code of an initial role of that agent (named a role-zero). In the following sub-sections, this change in abstraction level is highlighted more.

2.1 | Code under analysis

The systems under study were already introduced in the Chapter I: 6 | , there are 7 MAS selected with 23 agents, playing 89 roles (based on the previous assumption). Table IV-1 extend Table I-1 by showing the number of roles of each agent.

MAS	System	Agents	Roles	LOC	CC
1	Auction System -1-	2	5	322	29
2	Auction System -2-	3	7	397	47
3	Auction System -3-	3	15	525	79
4	Book Trading System	2	5	266	52
5	Hospital-Appointment Allocation System	2	11	520	100

MAS	System	Agents	Roles	LOC	CC
6	Home Automation System	9	42	1972	206
7	Treasure Hunt System	2	4	254	46
	Total	**23**	**89**	**4256**	**559**

Table IV-1 The MASs understudy

The test cases of these agents are the ones presented in the previous Chapter III: 5 | . For the dual-test-level code, and to ensure uniformity in analysing the relationship between agent properties, attributes, and testing levels; the test cases of the agent level are retained over those of the unit level in this part of the study. These dual cases are originally agents' communication codes; they are supposed to be tested at the agent (interaction) level.

Given that the retained test cases set was slightly changed, and to remove any potential concerns about its quality, the mutation testing technique (Sayward et al., 1978) was re-executed following the same strategy outlined in Chapter III: 5.1 | . Table IV-2 presents the number of test cases for each agent and the mutation test score at each testing level. The overall mutation score obtained was 98%, which confirms, again, the quality of the test cases and the rigour of their design strategy.

MAS	Agent	Nb Test Cases		Nb Mutations	Mutation Score
		Unit	Agent		
1	CarrierAgent	4	4	134	99%
1	CompanyAgent	4	4	182	99%
2	AuctionAgent	3	2	126	100%
2	BargainerAgent	7	1	144	99%
2	EnterpriseAgent	10	3	153	99%
3	Auctioneer	28	4	163	99%
3	BidderComp	6	4	69	99%
3	BidderHuman	15	5	73	99%
4	BookBuyerAgent	17	2	175	91%
4	BookSellerAgent	5	2	80	99%
5	HospitalAgent	7	4	215	99%
5	PatientAgent	23	5	391	98%
6	Building	5	2	88	97%
6	Bulb	16	5	311	95%
6	Controller	5	1	84	98%
6	LightSensor	10	6	358	96%
6	MeshNetGateway	8	2	86	95%
6	Room	8	2	175	97%
6	TempSensor	10	6	355	97%
6	ToggleSwitch	9	5	247	98%
6	Demo	4	0	112	93%
7	Player	17	1	99	100%
7	GameMaster	13	1	68	100%
	Total	**234**	**71**	**3888**	

Table IV-2: Number of Test Cases per Agent

2.2 | Independent variables

The independent variables for this study are the agent properties and attributes, and source code metrics, as defined in the JADE agent properties measurement model presented in the Chapter II: 4 | . These variables are categorised as follows:

- For Q1, Q2, Q7 and Q10: the 21 metrics (Table II-4).

- For the Q3, Q4 and Q8: the 14 attributes (Table II-7).
- For Q5, Q6 and Q9: the 7 properties Table II-6.

2.3 | Dependent variables

The dependent variables are test code and testing effort metrics. The set of metrics and the testing effort weighted sum function established in the previous chapter, as a part of the investigation of the optimal testing level for dual-test-level cases, can be used. However, to gain a more comprehensive understanding and deeper analysis of the testing effort, this set of metrics was refined. Rather than using a single metric to encompass various mocks, and another for different types of assertions and verification instructions, these two metrics were subdivided into more specific, atomic metrics, ensuring a finer granularity in measuring and analysing testing effort. The final adopted metric set is:

- TLOC: the number of test lines of code.
- NTC: the number of test cases.
- NbMock: the number of Mockito mocks.
- NbSpy: the number of Mockito spies.
- NbJatMock: the number of JAT agent mocks.
- NbAssert: the number of JUnit assertions called in the agent's test.
- NbMockAssert: the number of JUnit assertions in a JAT agent Mock.
- NbVerify: the number of Mockito verify instructions called in an agent's test.

To quantify the testing effort, the equation Eq.III-3 was updated with the new metrics. The Analytical Hierarchical Process (AHP) (Saaty, 1988) was again employed to guide the refinement of metrics and to determine the appropriate weight of each new atomic metric. The resulting weighted sum functions are presented in Eq. IV-1, and Eq. IV-2 . The consistency rate of the obtained weights is respectively 0 and 0.003, which are below the recommended threshold of 0.1 (Saaty, 1990). The overall new testing effort equation is presented in Eq. IV-3

$$\boldsymbol{NbAllAsserts = 0.333\ NbrAssert + 0.333\ \mathrm{NbMockAssert} + 0.333\ \mathrm{NbVerify}}$$

Eq. IV-1

$$\boldsymbol{NbAllMocks = 0.648\ \mathrm{NbJatMock} + 0.23\ \mathrm{NbSpy} + 0.122\ \mathrm{NbMock}}$$

Eq. IV-2

$$\boldsymbol{Test_{Effort} = 0.449\ TLOC + 0.235\ NTC + 0.152\ \mathrm{NbJatMock} + 0.054\ \mathrm{NbSpy} + 0.029\ \mathrm{NbMock} + 0.027\ NbrAssert + 0.027\ \mathrm{NbMockAssert} + 0.027\ \mathrm{NbVerify}}$$

Eq. IV-3

To summarize the dependent variables in this empirical study are:

- Q1, Q3 and Q5: the previous 8 test metrics.
- Q2, Q4 and Q6: 2 variables indicating the testing levels: unit and agent.
- Q7 to Q10: The testing effort function Eq. IV-3.

2.4 | Data collection procedure

After the theoretical framework was defined, two operational steps were taken to streamline the data collection procedures. First, setting the correspondence in between each agent's role code and its test cases. This involved annotating the test code with a special Java notation (@RUT), where each test method was marked with the name of the role (behaviour) being tested.

Secondly, the JMP software tool, proposed was upgraded for the second time to handle the shift of the abstraction level from agent to role-oriented when counting agent measurements. Slight modifications were made on the tool's results publication module, without affecting the original metrics measurement module. The JTF metric measurement process is already behaviour-centric, with metric values being counted and aggregated incrementally: starting from simple behaviours, then progressing to complex (top) behaviours, and ultimately considering the agent as a whole.

Once everything was set, the JMP tool was launched to collect the measurements. The tool generated two CSV files, one containing AO metrics of the 89 agents' roles under assessment (referred to as source code (SC) metrics in subsequent sections) and the other containing the test metrics per role, per test-level. Table IV-3 and Table IV-4 present some statistics on these metrics.

Metric	count	mean	std	min	25%	50%	75%	max
LOC	89	47.82	27.66	9	26	40	67	148
AdvSer	89	0.19	0.47	0	0	0	0	2
ReqSer	89	0.2	0.48	0	0	0	0	2
RecReqMsg	89	0.21	0.44	0	0	0	0	2
SentMsg	89	0.45	0.78	0	0	0	1	4
RecMsg	89	0.3	0.57	0	0	0	1	3
AgreeRate	89	0.14	0.31	0	0	0	0	1
NegoMsg	89	0.12	0.28	0	0	0	0	1.5
ExecMsg	89	0.19	0.49	0	0	0	0	2
RecExecMsg	89	0.04	0.11	0	0	0	0	0.5
LOC_COM	89	0.43	0.4	0	0.02	0.24	1	1
AddedRole	89	0.74	1.2	0	0	0	1	5
RoleComplexity	89	7.12	4.62	1	4	6	9	25
NSubBeh	89	0.11	0.55	0	0	0	0	3
BeliefSize	89	3.6	3	0	1	3	5	13
NbBeliefAccess	89	4.92	5.17	0	1	3	7	23
NbBeliefUpdates	89	1.75	2.14	0	0	1	2	8
SUC	89	0.33	0.42	0	0	0.2	0.54	2
NOM	89	2.64	1.82	1	1	2	4	9
AvgMtdPar	89	0.43	0.55	0	0	0	1	2.33
NbExp	89	1.01	1.36	0	0	1	2	8

Table IV-3 Descriptive statistics of source code metrics

Metric	count	mean	std	min	25%	50%	75%	max
NTC	89	3.42	2.78	1	1	2	5	15

TLOC	89	103.2	60.96	25	61	93	124	398
NbAssert	89	10.36	9.12	2	4	8	13	50
NbVerify	89	4.37	4.66	0	0	3	8	21
NbSpy	89	3.44	3.05	0	1	2	5	12
NbMokc	89	1.27	1.55	0	0	1	2	7
NbJatMock	89	1.57	1.6	0	0	1	3	5
NbMockAssert	89	3.62	5.87	0	0	0	6	28
Level_Agent	89	0.62	0.49	0	0	1	1	1
Level_Unit	89	0.74	0.44	0	0	1	1	1

Table IV-4 Descriptive statistics of test metrics

2.5 | Data analysis procedure

2.5.1 | Data pre-process

Initially, data from both JMP-generated CSV files were aggregated to create one dataset (shape 89*31), each row represents a pair <R, T>, comprising an agent's role source code metrics values and its test metrics values. The last two columns indicate whether the role was tested at the unit, agent or both levels (value 1 signifies that the role was tested at the corresponding level). Subsequently, the data underwent min-max normalization, a necessary step to reduce sensitivity to scale variability during the data analysis process.

The phases of logistic regression analysis and model prediction building necessitated additional pre-processing steps. These Phases involved binary classification models, requiring testing effort values to be categorized into two classes: low and high (0 and 1) (Ouellet & Badri, 2024; Moudache & Badri, 2022; Touré et al., 2018). To achieve this, a histogram chart with three bins was created (Figure IV-1) to visualize the data distribution and aid in determining the split value. It was decided to label the first category (testing effort ≤ 0.314), representing 79% of the sample, as the low-testing effort category (0), while the remaining values (testing effort > 0.314) were categorized as the high-testing effort category (1)

This testing effort categorisation resulted in an imbalanced dataset (80% vs 20%). To address potential classification inefficiencies, the SMOTE-Tomek Links technique (Batista et al., 2003) was employed to balance the dataset. The technique was chosen because it maintains features' spaces and does not duplicate the minority class. Moreover, it reduces class overlap by eliminating samples that could confuse the classifier.

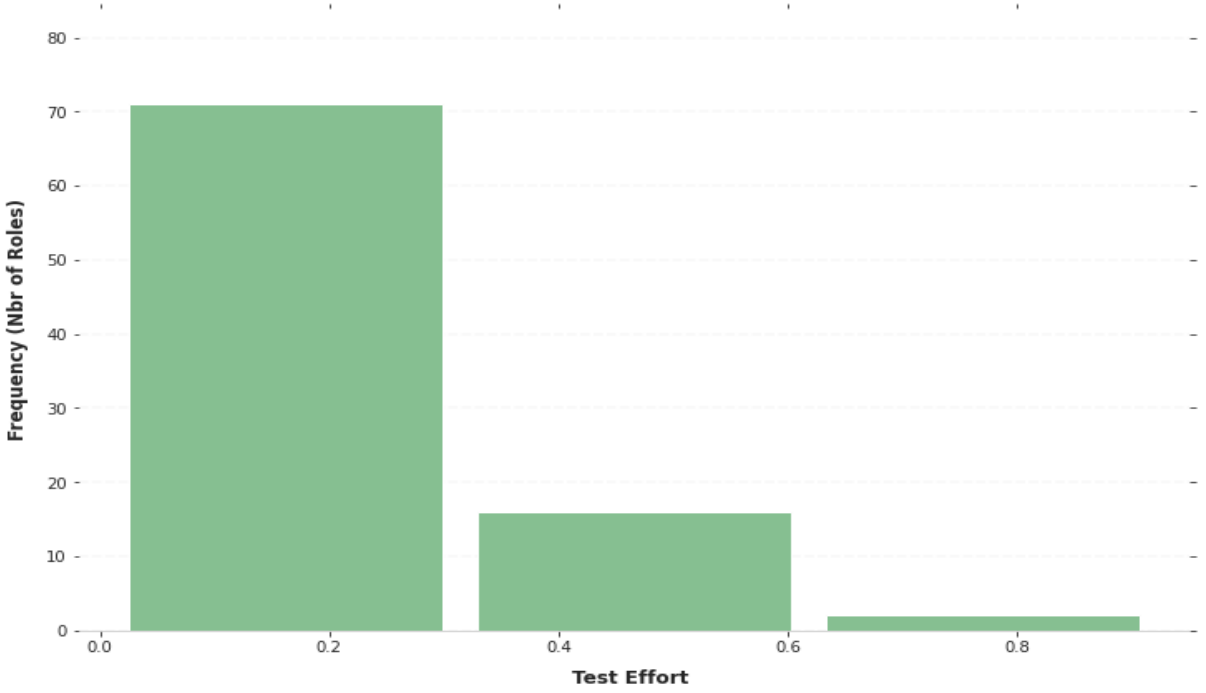


Figure IV-1 Histogram of agent testing effort distribution.

2.5.2 | Correlation analysis

Statistical correlation tests, were conducted to investigate research questions Q1 to Q3, exploring the relationships between test metrics (mt) and sets of agent's SC metrics (ms), attributes metrics (ma), and properties metrics (mp). The hypotheses for each question were as follows:

H1.0. There is no relationship between **a source code metric** (ms_i) and **a test metric** (mt_i).
H1.1. There is a relationship between **a source code metric** (ms_i) and **a test metric** (mt_i).

H2.0. There is no relationship between **a source code metric** (msi) and **a testing level** (li).
H2.1. There is a relationship between **a source code metric** (msi) and **a testing level** (li).

H3.0. There is no relationship between **an agent's attribute** (ma_i) and **a test metric** (mt_i).
H3.1. There is a relationship between **an agent's attribute** (ma_i) and **a test metric** (mt_i).

H4.0. There is no relationship between **an agent's attribute** (mai) and **a testing level** (li).
H4.1. There is a relationship between **an agent's attribute** (mai) and **a testing level** (li).

H5.0. There is no relationship between an **agent's property** (mp_i) and a **test metric** (mt_i).
H5.1. There is a relationship between an **agent's property** (mp_i) and a **test metric** (mt_i).

H6.0. There is no relationship between **an agent's property** (mpi) and **a testing level** (li).
H6.1. There is a relationship between **an agent's property** (mpi) and **a testing level** (li).

The Spearman rank correlation was used as a correlation test. This nonparametric test measures the strength and direction of the monotonic relationship between two variables without making assumptions about data distribution, unlike parametric techniques such as Pearson's correlation. The Spearman correlation coefficient (r-value) ranges from -1 to 1, with a positive (resp. negative) value indicating that the variables increase or decrease together (resp. opposite to each other). To determine significance (p-value), a threshold alpha of 0.05 was applied.

2.5.3 | Logistic regression analysis

In the second phase of the study, logistic regression (LR) analysis was employed to address questions 7 to 9. LR helps to understand the relationship between a dependent variable and one (univariate ULR) or more (multi-variants MLR) independent variables. The ULR was utilized to assess the individual impact of each source code metric, agent attribute, and agent property in predicting testing effort. While MLR was used to evaluate the combined effect of these elements. The purpose of this phase was not to develop the optimal prediction model but to explore and understand the relationship.

To determine whether a metric (attribute or property) significantly predicts testing effort, a threshold alpha of 0.05 was applied to the p-value of the regression coefficient for each metric (attribute or property). Further, to assess the models' performance, the metric AUR-ROC (Area Under the Receiver Operating Characteristics) is used, alongside the internal parameters of the LR model: b (estimated regression coefficient) and Pseudo-R2 (the ratio of the log-likelihood of the null model to that of the full model).

AUC-ROC is a widely used metric for model evaluation, combining sensitivity and specificity to effectively assess model performance while avoiding issues associated with subjective cut-off values. A higher AUC-ROC value indicates better model performance. A perfect prediction model scores an AUC-ROC of 1, while a score of 0.5 suggests a poor model. Generally, a good model will have an AUC-ROC greater than 0.75.

As well, the 10-fold cross-validation technique was employed as a model evaluation method to mitigate overfitting risks. Overfitting may occur since the models were constructed and tested using the same dataset, which is relatively small (only 89 samples). Also, in the case of SC metrics analysis, the number of dependent variables (21 SC metrics) is relatively large compared to the dataset size.

The 10-fold cross-validation technique involves partitioning the dataset into 10 subsamples, and repeatedly for 10 iterations, the model is built using 9 subsamples and evaluated with 1 subsample. The final model performance is determined by averaging the performance scores of the 10 iterations.

2.5.4 | Agent testing effort prediction model

In the third and final phase of this empirical study, machine learning techniques were employed to predict agent testing effort class (low or high) from SC metrics. Nine classification algorithms, widely used in the literature (Ouellet & Badri, 2024; Moudache & Badri, 2022; M. Badri et al., 2019; Vig & Kaur, 2018; M. Badri et al., 2017; Iwata et al., 2016; Satapathy et al., 2016; M. Badri & Toure, 2012; Braga et al., 2007) were employed: Logistic regression (LR), k-Nearest

Neighbors (kNN), Perceptron, DecisionTree (DT), Random Forest (RF), AdaBoost, Support Vector Machine (SVM), A Gaussian Naive Bayes (G-NB), Gradient Boost.

Building the best testing effort prediction model is beyond the scope of this study. However, to explore the feasibility of such a model and ensure confidence in the results a further pre-processing step of feature selection was undertaken. A multi-collinearity analysis was performed on the independent variables using the Variance Inflation Factor (VIF). This analysis revealed significant multi-collinearity among the variables, with the LOC metric showing the highest correlation, with a VIF score of 49. Consequently, the metric LOC was dropped from the dataset. Table IV-5 and Table IV-6 present the VIF scores before and after removing LOC. Despite the potential for further variable elimination (VIF scores still exceeding 10), it opted to refrain from that to avoid the loss of significant variables.

N	Metric	VIF
1	LOC	49.69
2	AdvSer	3.64
3	ReqSer	3.41
4	RecReqMsg	5.23
5	SentMsg	6.98
6	RecMsg	11.88
7	AgreeRate	4.09
8	NegoMsg	2.90
9	ExecMsg	3.92
10	RecExecMsg	10.16
11	LOC_COM	15.49
12	AddedRole	3.46
13	RoleComplexity	27.10
14	NSubBeh	2.37
15	BeliefSize	12.58
16	NbBeliefAccess	9.27
17	NbBeliefUpdates	13.40
18	SUC	4.66
19	NOM	11.89
20	AvgMtdPar	8.09
21	NbExp	4.95

Table IV-5 Independent variables variance inflation factor scores before dropping LOC

N	Variables	VIF
1	AdvSer	3.16
2	ReqSer	3.27
3	RecReqMsg	4.94
4	SentMsg	6.53
5	RecMsg	11.02
6	AgreeRate	4.09
7	NegoMsg	2.83
8	ExecMsg	3.77
9	RecExecMsg	9.64
10	LOC_COM	14.53
11	AddedRole	3.34
12	RoleComplexity	11.72
13	NSubBeh	2.29
14	BeliefSize	11.58
15	NbBeliefAccess	9.15
16	NbBeliefUpdates	13.07
17	SUC	4.66
18	NOM	11.68
19	AvgMtdPar	8.09
20	NbExp	4.66

Table IV-6 Independent variables variance inflation factor scores after dropping LOC

The nine models were assessed using the same methodology as in the previous phase, specifically employing 10-fold cross-validation with ROC-AUC as the primary evaluation metric. Since most of the selected machine learning algorithms require careful tuning of numerous hyperparameters[30], the Grid Search (GS) technique was applied for hyperparameter tuning. GS systematically explores various hyperparameter configurations to identify the one that maximizes the model's predictive performance (Belete & Huchaiah, 2022).

[30] External settings predefined before the learning process, they cannot be inferred from the data.

3 | Experiment results and discussion

3.1 | Correlation analysis

Figure IV-2, Figure IV-3, and Figure IV-4 summarize the results of the correlation analyses. They are heat maps of correlations coefficients of each of the: 210 pair-wise combinations of SC metrics and test metrics, 120 pair-wise combinations of agents' attributes and test metrics; and 70 pair-wise combinations of agents' properties and test metrics respectively. For clarity, only the correlated combinations are shown, with a p-value greater or equal to 0.05 (the null hypothesis has been rejected). In the following paragraphs, the absolute r-value is interpreted as: weak (≤ 0.3), moderate ($0.3 - 0.5$), or strong (> 0.5) relationship (Cohen, 2013).

3.1.1 | Correlation between SC metrics and both TM and testing levels

Figure IV-2 shows that in most cases there are correlations between SC metrics and both test metrics and testing levels, varying from positive to negative and ranging from strong to weak. Only a few cases show no relationships, for example the number of sub-behaviours a role have, the computed p-values where such low that the null hypothesis couldn't be rejected in most of the pairwise comparisons; the exception was between it and the number of spies, where a weak positive correlation is registered. This is likely because testing a role sub-behaviour at the unit level often requires spying on the agent (Chapter III: 5.2 |).

In Figure IV-2, 3 categories of SC metrics can be distinguished. First, the SC metrics related to agent internal structure size and complexity like LOC, belief size, access and manipulation metrics (belifSize, NbBeliefAccess, NbBeliefUpdates, SUC), role complexity, and the number of roles added ...etc. These metrics show a positive correlation, ranging from moderate to strong, with most of the test metrics. They tend to exhibit moderate positive correlations with the unit-testing, indicating that as these metrics increase, so do the size and complexity of the unit test cases. This is expected since the internal agent structure is tested mainly at the unit level rather than the agent level, where those metrics show a negative, moderate, correlations. It is important to note that these negative correlations do not suggest that an agent with a large and complex internal structure is less interactive. Instead, this finding should be interpreted considering that the data are for agent roles and not the agent as a whole; roles have more specific task-oriented behaviours. This observation is further supported by the subsequent category.

The second categories are associated with metrics related to agent interaction like the number of messages sent and received, the rate of communication level, the number of requests received, the agreement rate, etc. Unlike the first category, these metrics have a positive correlation, ranging from moderate to strong, with the agent-testing level and its associated test metrics: the number of JAT mocks and the number of assertions in the mocks. Furthermore, these metrics show

weak correlations and sometimes no correlations at all with metrics associated with unit-testing level like the number of assertions in the test method, the number of Mockito mocks and spies and the number of Mockito verify instructions. The data indicates that as the values of these interaction metrics increase, so do the size and complexity of agent-level test cases, necessitating more lines of test code and JAT mocks.

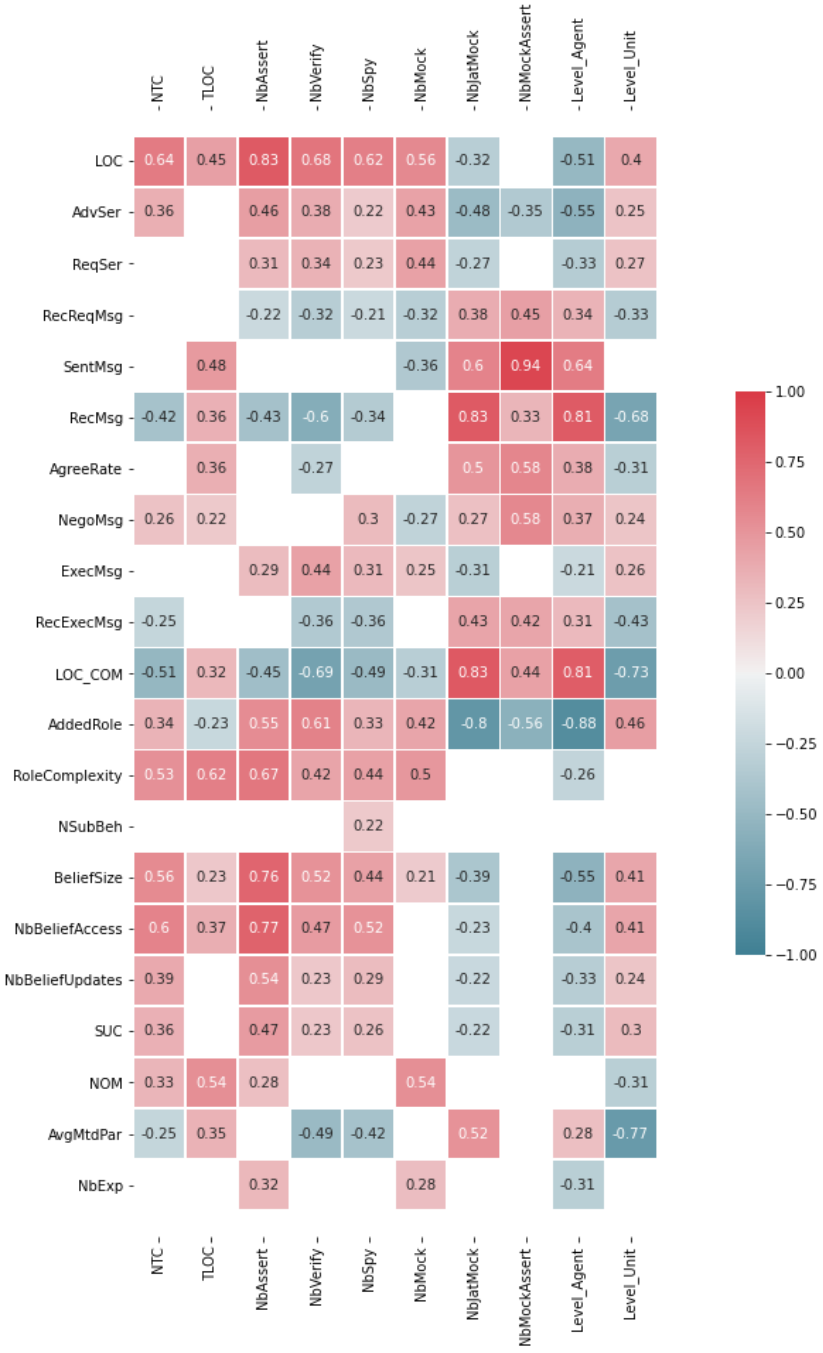


Figure IV-2: Correlation coefficient between Agent metrics and test Metrics

Interestingly, this category also includes the average method parameter metric, which is typically considered as a size metric in the OO paradigm (Xenos et al., 2000). The data revealed strong to moderate negative correlations with unit-testing level and its associated metrics, respectively. Occasionally, it showed positive, generally moderate correlations with agent-testing

level metrics, such as the number of JAT mocks and lines of test code. This can be attributed to the heavy use of methods in implementing the JADE interaction protocol[31]. In the adopted test strategy, these protocols were tested at the agent level using JAT mocks, rather than testing their methods separately.

Apart from the previous categories' metrics, three metrics do not follow the previous patterns namely the rate of negotiation message exchanged (NegoMsg), the number of methods (NOM) and the number of exception types (NbExp). NegoMsg has a positive correlation with both testing levels and with 5 of 8 testing metrics. This is because testing negotiation requires testing both the internal agent negotiation logic and the different interaction situations that may occur during a negotiation. The results indicate that as negotiation complexity increases, the size and complexity of test cases at both levels also rise, necessitating more test cases, lines of code, JAT mocks, and spies on the negotiation roles.

Regarding the number of methods, the results show positive moderate to strong correlations with the number of test cases, the number of test lines of code, the number of assertions in test method and the number of mocks. However, an unexpected negative moderate correlation was registered with the unit-testing level. This can be due to noise from the non-uniformity in testing strategies, particularly between methods within interaction protocols and ordinary methods. Notably, no correlations were found between this metric and the agent-testing level or its associated metrics (number of JAT mocks and assertions in mocks).

The number of exceptions metric exhibits relatively low correlations with test metrics, positively correlating with only two of the eight: the number of assertions and the number of mocks. Additionally, it shows a moderate negative correlation with the agent-testing level and no correlation with the unit-testing level.

Based on the above results, affirmative answers can be given to the research questions Q1 and Q2 which were:

Q 1. Is there a relationship between agent-oriented metrics and testing effort metrics?

Q 2. Is there a relationship between agent-oriented metrics and testing level?

The findings indicate that, in most cases, there are statistically significant correlations between AO SC metrics and both test metrics and testing levels. These correlations vary from strong to weak and can be either positive or negative.

[31] Jade provides a set of ready-made interaction protocols classes that developer can extend by overwriting only the methods responsible for preparing the messages to be sent or processing the received messages

3.1.2 | Correlation between agent attributes and both TM and testing levels

The correlation analysis results presented in Figure IV-3 suggest that there are correlations between agent attributes and testing metrics and testing levels in most of the cases, only 25% of pair comparisons are uncorrelated. These results reinforce previous findings: agent communication and interaction attributes are in strong and moderate correlations with the agent-testing level, respectively. This implies that as communication and interaction increase, so does the need for more test lines of code, JAT mocks, and assertions in the mocks. This aligns well with the adopted test strategy, as these two attributes are mainly tested at the agent level.

Attributes like agent structure size, perception, adaptation, learning, initiative, and function independence show moderate correlations with the unit-testing level along with moderate to strong positive correlations with this level metrics: the number of assertions in test methods, metric number of verify instructions, the number of spies and the number of mocks. Simultaneously, they exhibit moderate to strong negative correlations to the agent-testing level. The findings suggest that these attributes are tested at the unit level and the higher their values, the higher the size and complexity of unit test cases needed, and consequently, testing effort.

Another attribute that follows slightly the previous pattern, is behaviour complexity, this attribute registered most of the time moderate to strong positive correlations with most metrics except the number of JAT mocks and the number of assertions in agent JAT mock where no correlation is found, suggesting that these attributes can have a big impact on both the size and the complexity of test on both levels. The agent behaviour complexity can be related to both the inner agent logic complexity and its interaction and communication schema complexity. Still, no clear conclusion can be formulated on its correlation with the testing levels.

Regarding negotiation attributes, like its main metric NegoMsg, this attribute is respectively in a weak and moderate correlation with both testing levels. Agent negotiation is tested at both with little more emphasis at the agent level where it registered a strong positive correlation to the number of assertions in the JAT agent mock compared to no correlation with the number of assertions in test methods. The obtained data indicates that negotiation capability can have a slightly positive impact on test size, number of test cases and number of test lines of code to implement.

The attributes cooperation and reaction, show no correlations with the two testing levels. However, they show moderate correlations with the number of test lines of code, the number of assertions in JAT agent mocks and the number of assertions in test methods. Reaction presents also a weak positive correlation with the number of test cases and the number of spies. These findings suggest that while these two attributes may contribute to increased testing effort, it remains unclear at which level.

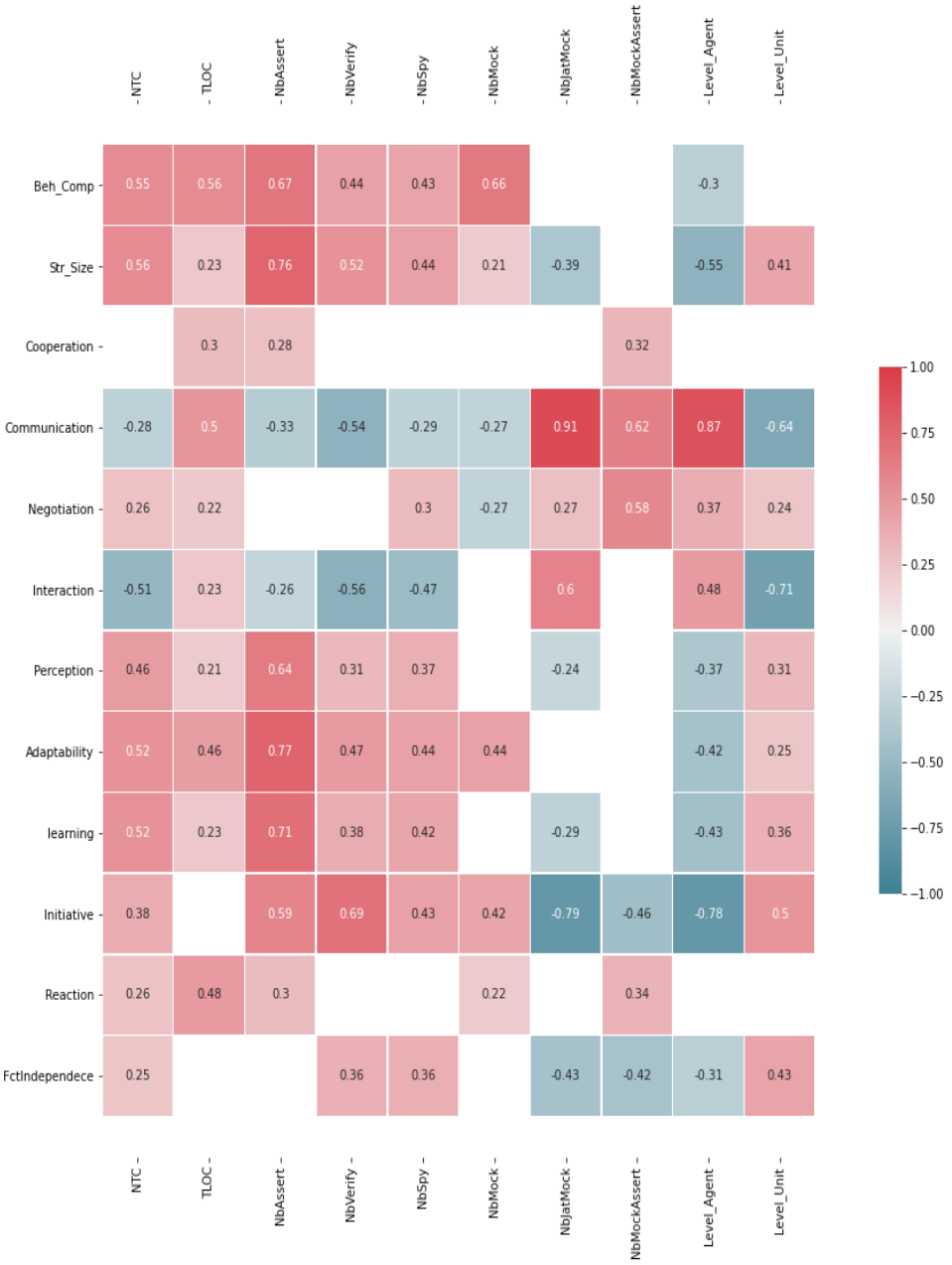


Figure IV-3: Correlation coefficient between Agents' attributes and Test metrics

From the previous data, positive affirmative can also be given to the research questions Q3 and Q4 which were:

Q 3. Is there a relationship between agent attributes and testing effort metrics?

Q 4. Is there a relationship between agent attributes and testing levels?

Overall, there are statistically significant correlations between agent attributes and both test metrics and testing levels. The exceptions are the attributes of reaction and cooperation, which did not show any correlation with the testing levels, despite exhibiting correlations with multiple testing effort metrics.

3.1.3 | Correlation between agent properties and both TM and testing levels

Figure IV-4 shows the correlation analysis results of agent properties. Only two of the seven properties (reactivity and intelligence) exhibit a low number of correlation relationships. The reactivity and intelligence are in moderate and strong correlations, respectively, with the number of test cases and the number of assertions in test methods. However, no correlation was found between these two properties and the testing level, suggesting that these properties have an impact only on the size of the test but no clear evidence on which level.

Regarding other properties, the sociability property shows strong positive correlations to agent-testing level and its associated metrics: the number of JAT agent mocks and the number of assertions in JAT mock. Sociability and its associated attributes are tested primarily at the agent level. The results suggest that a high sociability level leads to an increase in the number of test lines, the number of JAT agent mocks and assertions to implement. Whereas fewer test cases will be required at the unit level, as it is moderately negatively correlated with this level and its associated metrics.

The agent granularity, rationality and autonomy properties are in negative moderate to strong correlations with agent-testing level, and in weak to moderate correlations with agent unit. They present strong correlations with all the unit-testing level metrics. The obtained data suggest that an agent exhibiting high properties will require a high-testing effort.

The last property is proactivity, which presents a similar pattern as the previous metric set, it is in positive to moderate, correlations with most test metrics that are associated with the unit-testing level (except with NTC and TLOC). But unlike the previous metrics, no correlation was found with the unit-testing level and, at the same time, a moderate negative correlation is registered between the agent-testing level and its associated metrics. This property represents the ability of an agent to take initiative to achieve its goals, mainly linked to attributes tested at the unit level, which explains the results obtained.

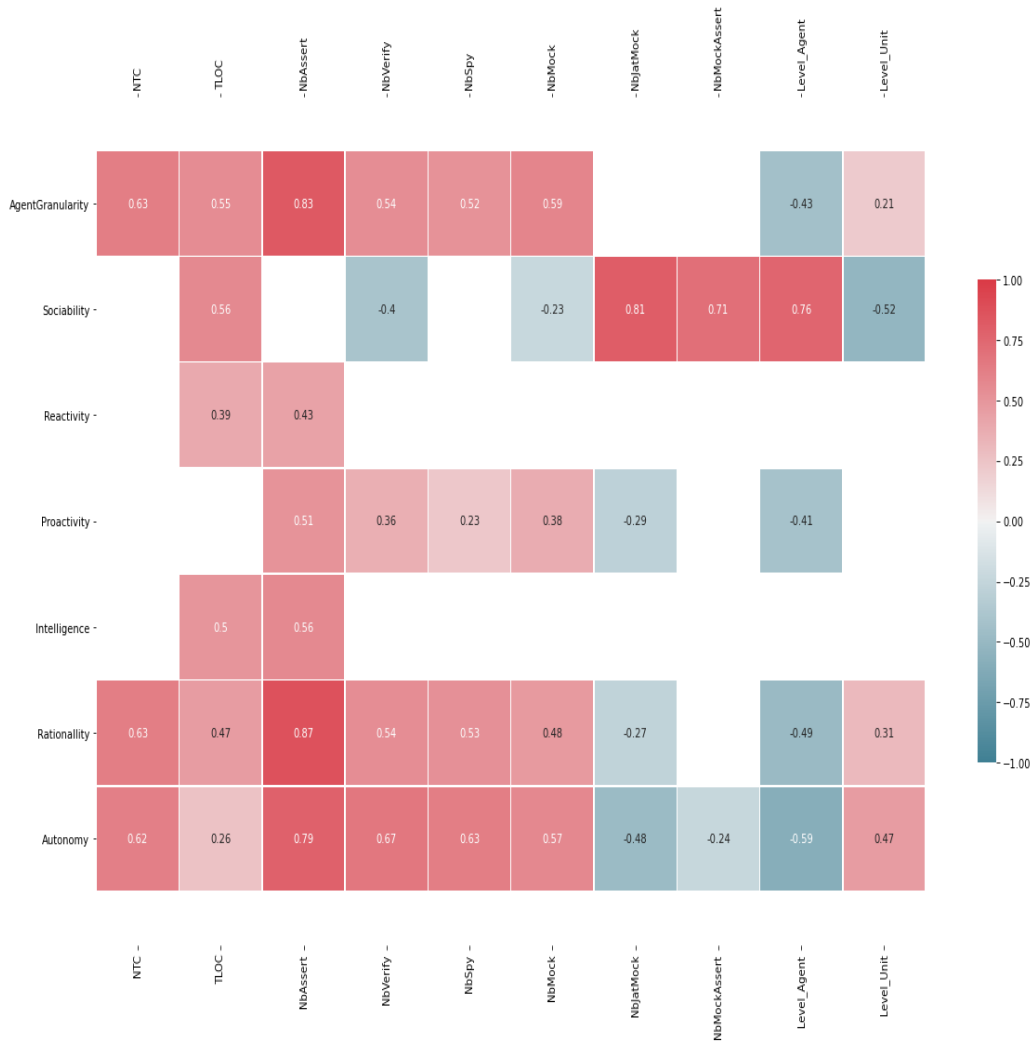


Figure IV-4: Correlation coefficients between Agents' properties and Test metrics

Based on the above finding, affirmative answers can be given to the research questions Q5 and Q6 which were:

Q 5. Is there a relationship between agent properties and testing effort metrics?

Q 6. Is there a relationship between agent properties and testing levels?

Overall, there are statistically significant correlations, generally positive, between agent property and test metrics and the testing levels. Notably, the properties of reactivity and intelligence did not show any correlation with the testing levels, despite having strong positive correlations with at least two of the testing effort metrics.

3.2 | **Logistic regression analysis**

In the next sections, to ease analysis, the LR results tables are sorted incrementally based on p-values.

3.2.1 | SC metrics LR analysis

3.2.1.1|ULR analysis of SC metrics and testing effort

From Table IV-7, only thirteen (13) of the obtained ULR models are significant, the b-coefficients associated with their metrics are high and significantly different from zero (p-value<0.05). According to R2 and ROC-AUC values, the model based on metric RoleComplexity, SentMsg and LOC are the most predictive of testing effort (ROC-AUC values are between 0.89 and 0.87). The other significant models' performance is weak, they scored moderate ROC-AUC values with a low R2 value.

For the remaining 8 metrics, the null hypothesis on their b-coefficients could not be rejected (P>0.05). Most of them show very low R2 values, such as LOC_Com ExecMsg, which scored the lowest b-coefficients and R2 value.

N°	Metric	Coef	Log-ll	R2	t	P>|t|	ROC AUC
1	LOC	8.3677	-65.9271	0.3302	6.1837	0.0000	0.8703 +/-0.13
2	SentMsg	6.4324	-66.0884	0.3286	6.1230	0.0000	0.8677 +/-0.12
3	NbBeliefAccess	4.4469	-83.9096	0.1475	4.5468	0.0000	0.7412 +/-0.16
4	RoleComplexity	10.7951	-56.7123	0.4238	6.0840	0.0000	0.8983 +/-0.10
5	NOM	3.4050	-89.5768	0.0899	3.8746	0.0001	0.7346 +/-0.17
6	NegoMsg	4.7961	-86.0805	0.1254	3.6772	0.0002	0.7615 +/-0.16
7	AddedRole	-3.9124	-91.0647	0.0748	-3.1552	0.0016	0.6459 +/-0.23
8	AgreeRate	1.8077	-93.5560	0.0495	2.8855	0.0039	0.6431 +/-0.09
9	BeliefSize	2.6979	-94.4976	0.0399	2.6742	0.0075	0.6661 +/-0.23
10	RecReqMsg	2.0968	-94.6387	0.0385	2.6118	0.0090	0.6375 +/-0.11
11	RecMsg	1.6539	-95.1344	0.0335	2.4744	0.0133	0.6779 +/-0.25
12	AdvSer	-2.8679	-94.5362	0.0395	-2.3610	0.0182	0.5531 +/-0.16
13	AvgMtdPar	1.5976	-95.6186	0.0285	2.3022	0.0213	0.6124 +/-0.15
14	NSubBeh	5.0010	-92.7267	0.0579	1.7605	0.0783	0.5714 +/-0.06
15	NbExp	1.9593	-96.7963	0.0166	1.7346	0.0828	0.6157 +/-0.13
16	RecExecMsg	1.1686	-97.2032	0.0124	1.5193	0.1287	0.5611 +/-0.09
17	SUC	-0.9761	-97.9087	0.0053	-1.0065	0.3142	0.3594 +/-0.23
18	ReqSer	0.7027	-98.0158	0.0042	0.8979	0.3693	0.6122 +/-0.11
19	NbBeliefUpdates	0.5885	-98.0504	0.0038	0.8635	0.3879	0.5957 +/-0.22
20	LOC_COM	0.3436	-98.1467	0.0028	0.7473	0.4549	0.3926 +/-0.30
21	ExecMsg	0.0938	-98.4194	0.0001	0.1223	0.9026	0.4798 +/-0.14

Table IV-7: ULR SC metrics Models

3.2.1.2|MLR analysis of SC metrics set and testing effort

Table IV-8 and Table IV-9 indicate that the built multivariate LR model is significant, even though the contributions of most metrics are not statistically significant. The model scored height performance in testing effort prediction. It has had high values of ROC_AUC and R2 (0.98 and 0.88 respectively), these metrics values are higher than those obtained when the SC metrics were considered individually in the ULRs. R2 value increased by 108% (compared to the highest obtained value in the previous analysis), showing that the combined effect of the metrics is more significant than when they were considered individually.

Model	Log-ll	R2	ROC AUC
LR(SC)	-11.325963	0.8849	0.9890 +/-0.03

Table IV-8 MRL SC model evaluation scores

N°	Metric	Coef	T	P>|t|
1	NOM	54.3140	2.0135	0.0441
2	SentMsg	62.0204	1.9965	0.0459
3	BeliefSize	-178.5854	-1.9503	0.0511
4	RoleComplexity	121.9318	1.9052	0.0568
5	NbBeliefUpdates	368.6240	1.8920	0.0585
6	AvgMtdPar	187.3289	1.8909	0.0586
7	LOC	-92.7277	-1.8833	0.0597
8	RecReqMsg	31.6323	1.8748	0.0608
9	AddedRole	-180.6136	-1.8635	0.0624
10	ReqSer	-158.5443	-1.8559	0.0635
11	SUC	-458.9859	-1.8421	0.0655
12	LOC_COM	-212.1396	-1.7538	0.0795
13	ExecMsg	123.6130	1.7056	0.0881
14	AdvSer	56.7119	1.6736	0.0942
15	NegoMsg	151.9786	1.5731	0.1157
16	NbBeliefAccess	37.6392	1.4995	0.1337
17	NbExp	-22.6154	-1.4364	0.1509
18	RecExecMsg	130.2023	1.0714	0.2840
19	AgreeRate	-35.3414	-0.9231	0.3560
20	RecMsg	4.0793	0.4300	0.6672
21	NSubBeh	204.5261	0.0000	1.0000

Table IV-9: MLR SC model's regression confessions

3.2.2 | Agent Attributes LR Analysis

3.2.2.1|Agent Attributes ULR analysis

The univariate analysis of 11 agent attributes are shown in Table IV-10 . Most of the models are significant, except function independence and interaction, their predictors' p-values are below the selected threshold and very low in performance metrics ROC-AUC and R2. Behaviour-complexity, communication and adaptability are the best testing effort predictive attributes. The ROC-AUC and R2 scores of their models are the highest. The reaction base model also scored a high ROC-AUC but its R2 is relatively low. The other statistically significant models had shown week prediction performance, their R2 scores are very low, and their ROC score is between 0.60 and 0.76.

N°	Attribute	Coef	Log-ll	R2	z	P>|z|	ROC AUC
1	Beh_Comp	12.3366	-69.8595	0.2902	5.8582	0.0000	0.8628 +/-0.16
2	Communication	6.7779	-73.8646	0.2495	5.8585	0.0000	0.8105 +/-0.16
3	Adaptability	10.0725	-74.1066	0.2471	5.7295	0.0000	0.8191 +/-0.15
4	Negotiation	4.5207	-87.8960	0.1070	3.4145	0.0006	0.7615 +/-0.17
5	Reaction	3.1393	-92.6961	0.0582	3.0978	0.0019	0.8163 +/-0.07
6	Cooperation	2.2577	-93.8587	0.0464	2.8335	0.0046	0.6176 +/-0.09
7	Initiative	-3.4920	-93.6682	0.0483	-2.8209	0.0048	0.6059 +/-0.22
8	learning	2.7958	-94.4731	0.0402	2.7043	0.0068	0.6582 +/-0.24
9	Str_Size	2.4725	-95.0306	0.0345	2.4996	0.0124	0.6559 +/-0.27
10	Perception	1.6726	-95.6574	0.0281	2.2867	0.0222	0.6389 +/-0.23
11	FctIndependence	-1.1686	-97.2032	0.0124	-1.5193	0.1287	0.5611 +/-0.09
12	Interaction	0.0430	-98.4259	0.0000	0.0439	0.9650	0.2270 +/-0.10

Table IV-10: ULR agent attributes Models

3.2.2.2|Agent Attributes MLR analysis

The Analysis of the combined effect of agent attributes resulted in a significant MLR model significant. Only 4 attributes had a statistically significant contribution. Surprisingly

FctIndependence and interaction are of them, which suggest that these attributes are better predictors when combined with others. The built model's prediction performance is excellent both R2 and ROC-AUC values are very high, 0.86 and 0.98 respectively. The R2 had increased by 196 % comparing with the ULR analysis, showing once again, that the score of the sum is higher than the individual ones.

Model	Log-ll	R2	ROC AUC
LR (Attributes)	-13.7901	0.8598	0.9842 +/-0.03

Table IV-11: MRL agent's attributes model's evaluation scores

N°	Attribute	Coef	z	P>|z|
1	Communication	41.0087	3.0188	0.0025
2	Interaction	-42.1998	-3.0131	0.0026
3	FctIndependence	-13.8972	-2.7277	0.0064
4	Adaptability	32.4616	2.0111	0.0443
5	Beh_Comp	36.5422	1.9271	0.0540
6	Initiative	-30.3796	-1.9022	0.0571
7	Reaction	-6.7873	-1.1487	0.2507
8	Str_Size	-11.7862	-0.3114	0.7555
9	Cooperation	-0.5608	-0.2023	0.8397
10	Negotiation	0.6725	0.1909	0.8486
11	learning	14.8155	0.0926	0.9262
12	Perception	2.6588	0.0297	0.9763

Table IV-12: MLR agent attributes model's regression confessions

3.2.3 | Agent properties LR analysis

3.2.3.1|Agent properties ULR analysis

Table IV-13 detailed the result of ULR of agent properties, most of the properties-based models are statistically significant, the exception was proactive-based models the null hypothesis on its b-coefficients cannot be rejected (P>0.05). Regarding the significant models' performances and based on ROC-AUC scores, sociability and agent granularity are the best testing effort predictive properties. They scored respectively of 0.86 and 0.85, followed by the rationality-based model which showed also a very good ROC-AUC score of 0.80, but an R2 slightly lower than the first two. The intelligence-based model ranks fourth with ROC AUC of 0.74. Reactivity and autonomy-based models come last, showing low ROC-AUC values and very low R2.

N°	Property	Coef	Log-ll	R2	z	P>|z|	ROC AUC
1	AgentGranularity	11.6381	-72.2319	0.2557	5.6352	0.0000	0.8531 +/-0.22
2	Sociability	9.0125	-70.0731	0.2779	5.7464	0.0000	0.8571 +/-0.11
3	Intelligence	12.7558	-80.6403	0.1690	4.9959	0.0000	0.7449 +/-0.14
4	Rationality	9.0424	-80.3380	0.1721	4.9085	0.0000	0.8000 +/-0.27
5	Autonomy	6.6832	-90.0449	0.0721	3.4984	0.0005	0.6755 +/-0.16
6	Reactivity	4.7989	-90.3654	0.0688	3.4322	0.0006	0.6939 +/-0.14
7	Proactivity	2.1573	-96.3672	0.0069	1.1532	0.2488	0.5796 +/-0.18

Table IV-13 ULR agent properties Models

3.2.3.2|Agent Properties MLR analysis

The result of analysis of the combined effect of agent properties shows (Table IV-14) that the resulting model is significant and has very good effectiveness in testing effort prediction with a high ROC-AUC value of 0.95 and R2 value of 0.60 (115% increased by comparing with the ULR analysis best result). However, in the built model only autonomy, and agent granularity properties show significant contribution coefficients.

Model	Log-ll	R2	ROC AUC
LR	-38.9818	0.5982	0.9490 +/-0.05

Table IV-14 MRL agent's properties model's evaluation scores

N°	Property	Coef	z	P>|z|
1	Autonomy	-14.9852	-4.0319	0.0001
2	AgentGranularity	40.9341	2.4743	0.0134
3	Rationality	-2.9165	-0.1870	0.8516
4	Sociability	19.3311	0.0000	1.0000
5	Reactivity	1.3665	0.0000	1.0000
6	Proactivity	-15.0619	-0.0000	1.0000
7	Intelligence	-23.7419	-0.0000	1.0000

Table IV-15 MLR agent properties model's regression confessions

3.3 | **Agent testing effort prediction models**

The evaluation results of the built prediction models are present in Table IV-16. All models scored excellent, very high values on most of the metrics, suggesting excellent abilities to separate testing effort classes, the median values for ROC-AUC and R2 are 0.98 and 0.95 respectively. The best-performing Model is Gradient Boosting with the highest scores for both ROC-AUC (0.99) and accuracy (98%). It has the lowest error rates. Followed by AdaBoost with nearly the same ROC-AUC, but with a slightly higher false positive rate. The other models LR, Random Forrest, SVM and Perceptron are strong predictors also with ROC-AUC equal to 0.98, they offer reliable predictions with minimal errors.

Gaussian Naive Bayes is the weakest model in this analysis. It has the lowest accuracy, precision, and recall, and the highest false positive and false negative rates. Its ROC-AUC score is also the lowest, indicating poor performance in distinguishing between low- and high-testing effort. This model is less reliable compared to the others, likely because its assumption that features are conditionally independent, which is not the true for current dataset.

The obtained results confirmed that agent testing effort can be highly predicted form its source code.

N	Model	Accuracy	Precision	Recall	FPR	FNR	F1-Score	ROC AUC	R2
1	GradientBoost	98%	98%	99%	3%	1%	98%	0.9918	0.98
2	AdaBoost	94%	93%	99%	10%	1%	95%	0.9877	0.94
3	LR	95%	93%	99%	9%	1%	95%	0.9857	0.95
4	Random Forrest	95%	97%	93%	3%	7%	95%	0.9795	0.95
5	SVM	94%	92%	97%	10%	3%	94%	0.9795	0.93
6	Perceptron	93%	92%	94%	9%	6%	93%	0.9714	0.93

7	DecisionTree	96%	95%	97%	6%	3%	96%	0.9571	0.96
8	KNN	95%	95%	96%	6%	4%	95%	0.95	0.95
9	Gaussian NB	86%	87%	89%	16%	11%	87%	0.9224	0.86
			Median				**95%**	**0.98**	**0.95**

Table IV-16: Prediction models evaluation results

Based on the previous table, the answer to the last research question: *Could agent **testing effort** be predicted from **SC metrics***? is: Yes, agent testing effort can be predicted to a great extent from its source code.

4 | Threats to validity

Scientific studies are open to validity threats and limitations, and the work within this thesis makes no exception. The study's findings should be viewed as exploratory and indicative rather than conclusive.

4.1 | Generalisation validity Threats

The main threat that may limit the generalisation of the finding of this study is the size of sample of agents under study, only 23 agents, which is small. This issue has been known since the beginning of the research, the lack of benchmarks and open industrial systems were and are still known issues in the field of multi-agent systems (Mascardi et al., 2019; Dix et al., 2012).

Nonetheless, efforts were made throughout the study to mitigate this threat in various ways. The sample was composed of a diverse range of agent types (autonomous, subordinate, reactive, proactive) with different sizes and complexities. Also, shifting the abstraction level from agent to agent's role, in the help to introduce greater diversity into the collected data. Besides, techniques such as dataset balancing, k-fold cross-validation, and feature engineering were employed to reduce the risk of model overfitting and to build confidence in the results of the machine learning phases

Also, it is believed that the changing of abstraction level from agent to agent's role helped to introduce more diversity to the collected data. Furthermore, techniques like dataset balancing, k-fold cross validation and feature engineering were applied to reduce the risk of model overfitting and subsequently to build confidence in the result of machine learning phases.

4.2 | Internal validity threats

Internal validity threats were the primary concern of the research team and were acknowledged and addressed from the beginning of the study. These threats are mainly associated with the JADE agent measurement model and test cases. The measurement model was developed specifically for this study and validated on the same systems used in the testability investigation, which could introduce potential bias. However, the well-structured and thoroughly documented approach provided by UFAPM (Chapter II:) minimized this threat. It is deemed less risky to the study than using existing models, many of which are either unclear or incomplete.

Regarding the test cases, they were also controlled risks. They were developed following a rigorous and uniform test design strategy, the McCabe technique with a completion criterion of 100% coverage. McCabe promotes analytical over arbitrary test case design every statement in the agent code is covered at least once. Furthermore, their quality was validated using the mutation test technique, and despite the strengths of the retained mutators list (0.89 mutations per LOC), the average mutation score was 98%.

Another internal threat is associated with the correspondence between source code and test code. This correspondence was established by manually annotating each test method/case with the role under test, using a special Java notation (@RUT). This task was performed by the D.Sc candidate, an experienced software developer in JADE agents, Java language and Java testing. To mitigate the risk of human error, all annotations were reviewed for potential errors three months later by the same individual. Additionally, since the development of the test cases, particularly agent-level test cases, was originally behaviour-centric (Chapter III: 5.2 |), it facilitated the identification and annotation of test cases.

4.3 | Construct validity threats

This type of threats was mitigated via using of automatic software tools and scripts to collect and process the data all along the different phases of the study. The JMP tool was used to analyse the code, identify the agents' roles and their test cases, and calculate both source code and test metrics (Chapters II to IV). The PIT Tool was used for mutation technique execution (Chapters III and IV). XLSTAT for statistical test evaluation (Chapter III). Python scripts for processing the collected data (Chapters II to IV).

4.4 | Conclusion validity threats

The validities of the conclusions were confirmed by empirically answering each research questions of the study (Chapter II to IV). For the main research questions, the significance and strength of each identified relationship between agent source code metrics (attributes, properties) and testing effort metrics (test-levels) were statistically demonstrated (correlation test). The effectiveness of agent properties, attributes and source code metrics in predicting the testing effort, were statistically measured (b-coef, R2, ROC-AUC) and proved (P-test). The prediction models' qualities were demonstrated using the robust ROC-AUC metric, and the 10-fold cross-validation

5 | Conclusion

This chapter delves into the core study of this thesis, aiming to understand how an agent's properties and attributes impact its testing effort at both the unit and agent (interaction) levels and to explore the feasibility of predicting an agent's testing effort from its source code.

Utilising the outcomes of the previous two chapters, the JADE measurement agent model and JTF base test case, the 23 JADE agents source code and their developed test code, were statically analysed. This analysis measured these agents' properties, attributes, and testing efforts. Then the collected data underwent a three-step statistical analysis process

The Correlation analysis was used, in the first step, to examine relationships between agents' properties, attributes, and source code (SC) metrics on one side, and testing effort metrics and testing levels on the other. The analysis revealed that, in many cases, there are statistically significant correlations, ranging from strong to weak and from positive to negative.

For instance, agent granularity, autonomy, rationality, and their related attributes like agent structure size, adaptability and learning show moderate to strong positive correlations with testing effort at the unit level, indicating that agents with higher values for these properties and attributes require more effort for unit-level testing. Similarly, the sociability property and its related attributes like communication and negotiation exhibit moderate to strong positive correlations with testing effort at the interaction level; indicating that sociable agents with higher values for these attributes require more effort for agent-level testing.

In the second step, logistic regression techniques (both univariate and multi-variant) were utilised to examine the individual and combined effect of SC metrics, agent attributes and properties, on the agent testing effort. The finding bolstered those of the first step and reinforced the confidence in the feasibility of predicting agent testing effort from source code metrics, as the majority of the linear regression models proved to be statistically significant.

In the final step, machine learning techniques were employed to develop exploratory binary classification prediction models for agent testing efforts. The models were assessed using 10-fold cross-validation and ROC-AUC metrics. The results yielded high-quality prediction models, all achieving ROC-AUC scores above 0.86. The best-performing Model is Gradient Boosting with the highest scores for both ROC-AUC (0.99) and accuracy (98%). This provided robust evidence to conclude that agent testing effort can be accurately predicted from its source code.

Conclusions and Future Works

1 | Conclusion

Multi-agent system quality remains a significant barrier to the widespread adoption of agent technology in the industry. Despite being a crucial quality assurance component, agent testing is notoriously difficult. Our literature review revealed that few studies have examined the reasons behind this challenge, and those that did, provided only superficial insights.

Recognising the urgent need to understand agent testability, we conducted a comprehensive empirical investigation into JADE agent testability through this work. The testability was examined from the perspective of agent testing writing effort at both **unit** and **agent** (interaction) testing levels. Agent source codes were statically analysed, and measurements of more than 30 metrics were collected and processed using statistical methods and machine learning techniques to answer the research questions. The key findings were:

- **Relationships identification**: Identifying the relationships between agent properties/attributes and testing effort/testing levels, highlighting which property/attribute affects testing and on which level.
- **Impact of properties/attributes**: Gaining detailed insights into how both individually and collectively agent properties/attributes impact the complexity and effort involved in testing at different levels.
- **Testing effort predictive models**: Successfully developing highly accurate experimental predictive models for testing effort using advanced machine learning techniques such as Gradient Boosting and Random Forest.

Throughout the journey of answering the research questions, numerous theoretical and technical challenges arose. The absence of a complete and effective measurement model and testing framework for JADE agents, coupled with the scarcity of open-source JADE agent systems, posed significant obstacles. However, we confronted these challenges head-on, and this perseverance led to many contributions in the field of MAS quality in general, and agent testing in particular, even before achieving the main research goal

- **A Unified Framework for Agent Properties Measurement (UFAPM):** a novel framework is proposed to systematically define and measure agent properties. Using the Goal-Question-Metric (GQM) approach, the framework for tailoring a context-specific agent measurement model.
- **A JADE Agent Measurement Model:** a complete, effective, and robust model to evaluate JADE agent properties, and attributes using source code metrics.

- **JADE Testing Framework (JTF):** a comprehensive testing solution for JADE agents, testing at both unit and agent interaction levels.
- **JADE Agent Testing Effort Measurement Model:** focusing on measuring agent test writing effort directly from the test code.
- **JADE Measurement Project Tool (JMP):** a code analyser software tool for JADE agents, providing a practical and automated way to measure more than 25 agent source code metrics and 8 testing effort metrics.
- **Benchmark of Agent Tests**: over 431 high-quality test cases for 23 agents are available with open access on GitHub, which can serve as a benchmark for other studies.

Whereas the main research contributions are:

- **Empirical Study on Agent Testability**: an in-depth empirical investigation advancing the understanding of agent properties, attributes, and their relationships with agent testing effort. This study offers both theoretical insights and practical tools that set the stage for more efficient testing strategies, enhancing the overall quality and reliability of multi-agent systems.
- **Predictive Models for Agent Testing effort**: ready to use and high-quality JADE agent testing effort prediction models required for agents in future MAS projects

2 | Future works

2.1 | Agent testability

As mentioned previously, the findings in this study should be viewed as exploratory and indicative rather than conclusive; thus, further investigations are required. Looking ahead, we intend to replicate the study on a larger number of more complex, and ideally industrial multi-agent systems.

Another potential area for future research is the investigation of inter-source-code metrics relationships. In this study, we found relationships between these metrics, and we attempted to mitigate their effects using multi-collinearity analysis. Studying these relationships will allow us to identify the best representative subset of source-code metrics, from which the most efficient prediction model (high performance, with fewer features to process) can be built.

Additionally, we aim to predict testing effort in a quantifiable manner (continuous value), rather than just binary classification (high, low), using machine learning regression techniques.

2.2 | The UFAPM and JADE agent measurement model

We plan to enhance them by addressing the limitations identified in the threats to validity section of Chapter II: Chapter II: 7 | We aim to develop a software tool to assist users of the UFAPM in process execution, which will help minimize user errors, enforce guideline applications, and automate candidate metrics identification. Additionally, we intend to migrate the Attributes/Metrics repository into a more manageable structure, such as a database.

Furthermore, further study is necessary to validate the framework on other agent-developing platforms that support different agent definitions (such as JACK, Jadex, Madkit) and on large, real-world MAS.

Additionally, we plan to conduct another investigation into inter-metric relationships to enhance the framework by a metric selection decision support mechanism. This mechanism will be helping to identify the optimal set of representative metrics, which significantly reducing the time and effort required for defining and executing agent measurement models.

Regarding the JADE agent properties measurement model, we plan to enhance it by incorporating the executive messages from the agent's GUI in the autonomy measurement; using dynamic metrics to gain more insights into certain agent properties like reactivity and proactivity; and revalidating the model on a larger sample of JADE multi-agent systems.

2.3 | JADE agent Testing Framework

Additionally, alongside planning to conduct further validity studies of JTF on other MAS projects, we intend to refine it and enhance its applicability and efficiency in agent testing by:

- Integrating integration-test level,
- Addressing the issue of flaky tests by incorporating libraries like ConcurrentUnit[32] or Awaitility[33] to better handle synchronisation between JTF's test thread and the AUT's thread,
- Enrichment of JTF DSL by Adding more JADE-test-oriented instructions and assertions.
- Supporting BDI agents, by integrating JAT4BDI,

These improvements will significantly contribute to refining JTF and enhancing its applicability in various MAS projects, ensuring higher quality and efficiency in agent testing.

[32] https://github.com/jhalterman/concurrentunit
[33] http://www.awaitility.org

Appendices

Appendix -1-: Metrics List

N°	Metric	Description
	ACB	The Average Complexity of Behaviours: average cyclomatic complexity of agent behaviours.
	ACM	Agent Competence Metric: the effectiveness of an Agent.
	ACR*	Architectural Complexity Rank (subjective):3 types of architectures: rule-based, goal-based, IA-based.
	ActMax*	Maximisation of success: the capacity to maximize the expected result of the actions
	AGA	Agent Goals Achievement : AGA = logk+1(G+1) G number of goals achieved by the agent during execution, k max possible Number of goals to be achieved by the agent.
	AgEffAcq-Perc*	Agent's ability to use sensors to perceive the relevant components and data in the environment
	AgType*	Agent type : simple (model based) or goal (utility) based
	AHF	Attributes Hiding Factors: number of private variables
	AL	Autonomy Level: the measure of the set of goals G multiply by the measure of the set of uncertainties.
	AMS	average sent message size.
	ANSB	The Average Number of Scheduled Behaviours in each moment.
	AOC	Agent Operations Complexity: mean complexity of operations to be performed to achieve goals.
	APM	Action Plasticity Metric: number of possible actions of the agent.
	ARA	the Average number of Roles played by Agent
	ASAd	Agent Services Advertised: number of services that agent advertise.
	ASG	Agent Structure Granularity (for composite variable): ASG = $\sum$ Ni /k-1 where a variable is saw as a tree: k is the number of the nodes in the tree, Ni is the height of the i^{th} node.
	ASKM	Agent Skill Metric: agent performance in different situations.
	ASM	Agent Support Metric: the number of agents communicating to complete the task.
	ASM	Agent Shift Metric: switching time taken by an agent moving from one environment to another.
	AVM	Agent Versatility Metric: the adaptability of an agent to different environments.
	BC	Behaviour Complexity: the Complexity of the services offered by agent.
	BG	Behaviour granularity: The agent's behaviours complexity
	BR	Overloader Role metric: the amount of agent's sent messages comparing it to the amount of sent messages by the agents playing the same role
	BRscore	Whited sum function of the amount of processed perception data and the belief update rate.
	BSA	The Behavioural Size of the Agent: number of agent behaviours.
	CAM	Cooperation Agent Metric: the amount of communication among agents and the help given by one agent to another.
	CCM	Communicative cohesion metrics (CCM): the ratio of internal relationships (inter-actions) to the total number of relationship
	ComAutAb*	Agent's ability to undertake autonomously communication with other agent
	ComMin*	The ability of agent to carry out tasks or goals with minimal communication.
	Communication	Sum of incoming outgoing communication
	CompGrad*	The degree of competition between the agent and the other MAS agents.
	CoopGrad*	The degree of cooperation between the agent and the other MAS agents.
	CorrChan-Reac*	Correlation between agent reaction and environment change.
	CP	Capacity of Perception (CP): The number of environment's objects an agent can perceive divided by the total number of environment's objects.
	DAA	The number of strategies that agent can adopt to the total number of possible strategies.
	DAS	Dynamism in Agent Structure (for dynamic variable): the difference between agent structure size in two instances of time.
	DCA	The ratio of solutions identified by the agent to the total number of possible solutions.
	DDA	The ratio of quality of agent identified solution to user expected quality.
	DefBeh*	Is the agent's reaction were pre-established by the designer or not?
	DIA	degree of individual autonomy: rate of agent performance when it acts autonomously to its performance under user's supervision.
	DiaErPrAb*	Agent's ability to diagnose errors and problems during execution.
	DNG	Degree of the novelty of a goal: this metric represents the difference between the new goal and the current one. It is count as the distance between goals.

N°	Metric	Description
	DPA	The ratio of the efficiency of agent's choice to the maximum possible efficiency the agent would achieve if he knew the user's preferences.
	EE	Emergence efficiency: The Ratio of the number of utile emergence situations to the total of emergence situations.
	EHF	Exception Handling Functionality: number of exceptions handled by Agent
	EMR	Executive Messages Ratio: ratio of executive messages received to all the received messages. EMR = 1− (ME/MR)
	EurFinAb*	Effectiveness in funding suitable heuristics to achieve goals.
	ExcMan-Ab*	Effectiveness in managing exceptions.
	FBC	Frequency of behaviour changing: The number of executed instructions that change the agent behaviour per a unit of time (Δt)
	FCG	Frequency of creating new goals: The number of executed instructions to create new goals during a time unit (Δt).
	FEC	Frequency of environment changing: The number of executed instructions to change the environment's objects by a unit of time (Δt).
	FP	Frequency of Perception: The number of instructions an agent executed to perceive the environment in a unit of time (Δt).
	FRC	Frequency of role changing: The number of times an agent changes his role in a unit of time(Δt).
	FRsC	Frequency of relationship changing: The number of specific instructions that lead to a changing in the relationship between roles, executed by an agent in a unit of time (Δt).
	FSC	Frequency of structure changing: The number of executed instructions that change the agent's structure per unit of time (Δt).
	FSU	Frequency of State Update: number of statements that accesses and modifies variables during execution.
	GAA	Goal Achievement Acceleration: measure the optimisation level of the agent progression to reach its goals according to the time
	GPscore	the average fraction between the tasks the agent was able to decompose out of a goal and all possible tasks related to the goal.
	GUR	Goal achievement by using resources: the efficiency of agent in the realisation of his goals (time and resources consumption)
	IM	Incoming Message: number of incoming Messages.
	insMod*	Does the agent possess an internal model of the actions and intentions of other agents?
	KUG	Knowledge Usage: Average of agent beliefs used in decision statements
	KUP (SUC)	Knowledge Update (State Update Capacity): Number of statements that update agent internal state (beliefs /variables).
	LearAb*	Learnability: the ability of agent to learn
	LScore	Whited sum function of the amount of learned Information the update rate of the information.
	MAG	Messages to achieve the goals: the rate of executive Messages sent divided by the number of goals achieved.
	MbReq	Messages by a Requested Service: number of messages exchanged by an agent doing the negotiation when another agent is requesting a service from him.
	MC2	Methods per Class (MC): number of implemented public methods or offered services.
	MoreRol*	Can the agent play multiple roles?
	MoreRol2	Agent roles (seen as cohesion): the number of internal dependencies divided by the number of possible internal dependencies.
	MR	Overloaded Role metric: measures the amount of an agent's received messages comparing it to the number of received messages by the agents playing the same Role.
	MtReq	Messages Sent to Request a service: number of messages exchanged by the agent doing the negotiation when it is requesting a service from another agent.
	Nci	The number of constraints defined for agent .
	NCR	Number of Cognitive Rules: number of actions that affect the internal beliefs or state of the agent.
	NegAg*	Agent's ability to negotiate task assignments.
	NegAg2	Negotiation: Average of communication
	Nei	The number of events on which agent will respond.
	NIICi	The number of Inbound Interactions from Cooperative events for agent .
	NIIIi	The number of Inbound Interactions from Independent events for agent .
	NIIIi	The number of Inbound Interactions for agent : NIIi = NIIIi + NIICi.
	NIP	The number of Interaction Protocols an agent implement
	Nki	The number of knowledge the agent have.

N°	Metric	Description
	NOG	The number of goals.
	NOIi	The number of Outbound Interactions for agent.
	NOR	The number of roles.
	Npi	The number of properties defined in the agent .
	NPR	The number of Processed Requests: the number of processed requests divided by the number of received requests.
	Nresi	The number of resources the agent will handle.
	Nri	The number of roles may play by an agent.
	NRM	The number of Response Messages: The number of possible response messages divided by the total number of received messages.
	Nsi	The number of services the agent can provide.
	OM	Outgoing Message: number of outgoing Messages.
	PAscore	Weighted sum function of the agent perceptions and actions adversity.
	PercQual*	Effectiveness of agent perceptions processes.
	PlaConst*	Plan construction ability: is agent able to build a plans
	PosStr*	Does the agent occupy a subordinate position in structure or not?
	R_a	Agent's relative decision making power: the number of votes agent can cast; on the way to pursue a goal, divided by the total number of votes that can be cast.
	RAP	Rate of Achieved purpose: The ratio of behaviours that achieved their goals to the number of executed behaviours.
	RFM	Response For Message: average of External and internal calls invoked in response to received messages.
	RightRol*	Agents change roles during problem resolution according to changes in the environment?
	RII	The ratio of interaction intention: number of situations where agent has the intention to cooperate to the total number of cooperation situations.
	RIU	Rate Of Interaction Utility: The ratio of the interaction situations in which the agents achieve their goals and the whole interaction situations
	RLRSs	The ratio of the Lack of Requesting Services: the number of failed services Requested divided by the number of executed behaviours.
	RPA	The Ratio of Plans per goal an agent have.
	RRA	The ratio of Resources Availability: The number of available resources divided by the number of necessary resources.
	RRG	The Ratio of Reached Goal: The ratio of reached goal compared to the total number of triggered ones.
	RRM	Resource Receptive Metric: the resources accessible in other environments.
	RSC	Rate of agent State Change: The ratio of agent's behaviours broken before reaching its purpose to the total number of executed behaviours
	RT (TRC)	Response Time (Time to respond to changes): The average time an agent took to react to an event.
	RU	The ratio of understand-ability: The ratio of messages understood to the total number of all received messages.
	SAS	Size of Agent Structure: number Attribute or variable represent agent internal structure.
	SC (KG)	Structural Complexity (Knowledge Granularity): The complexity of Agent knowledge structure.
	SD	Social Dependence: rate of tasks dependent upon others (relative to all agent tasks).
	SharTask*	Does the agent share tasks with another agent or not?
	SharTask2	Agent ability to share tasks with other agents (seen as coupling): The number of external dependencies divided by the number of possible external dependencies
	SI	Social Integrity: is the agent's softest spot or resistance to external influence
	SPK	Size of Procedural Knowledge: number internal beliefs or stats of agent.
	SRR	Services Requests Rejected by the agent: rate of rejected services request.
	TAP	Time to Achieve Purpose: the average execution time to achieve the goals.
	TM	Trust Metric (TM) the trust factor of an agent and its environment.
	Tr&RepMod*	The need to use trust and reputation models.
	UF*	Adjusted Function Point: weighted sum function of parameters: external inputs, external outputs, external inquiries, external interfaces, internal data structures, algorithmic complexity and knowledge complexity factor.
	VD (ISS)	Variable Density (Internal State Size): Somme of agent variables size (by byte).
	WMC (MC)	Wight Method per Class: Somme of cyclomatic complexity of agent method.

Bibliography

Abdullah, D., Khan, M. H., & Srivastava, R. (2015). Flexibility: A key factor to testability. *International Journal of Software Engineering & Applications (IJSEA)*, *6*(1), 89–99.

Achtaich, A., Roudies, O., Souissi, N., Salinesi, C., & Mazo, R. (2019). Evaluation of the State-Constraint Transition Modelling Language: A Goal Question Metric Approach. *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B*, 106–113. https://doi.org/10.1145/3307630.3342417

Albattah, W. (2022). Software Package Testability Prediction Using Object-Oriented Cohesion Metrics. *2022 13th International Conference on Information and Communication Systems (ICICS)*, 155–161. https://doi.org/10.1109/ICICS55353.2022.9811192

Alonso, F., Fuertes, J. L., Martínez, L., & Soza, H. (2008). Measuring the Social Ability of Software Agents. *2008 Sixth International Conference on Software Engineering Research, Management and Applications*, 3–10. https://doi.org/10.1109/SERA.2008.32

Alonso, F., Fuertes, J. L., Martínez, L., & Soza, H. (2009). Towards a set of Measures for Evaluating Software Agent Autonomy. *8th Mexican International Conference on Artificial Intelligence - Proceedings of the Special Session, MICAI 2009*, 73–78. https://doi.org/10.1109/MICAI.2009.15

Alonso, F., Fuertes, J. L., Martínez, L., & Soza, H. (2010). Evaluating Software Agent Quality: Measuring Social Ability and Autonomy. In T. Sobh & K. Elleithy (Eds.), *Innovations in Computing Sciences and Software Engineering* (pp. 301–306). Springer Netherlands. https://doi.org/10.1007/978-90-481-9112-3_51

Alonso, F., Fuertes, J. L., Martinez, L., & Soza, H. (2010). Measuring the Pro-Activity of Software Agents. *2010 Fifth International Conference on Software Engineering Advances*, 319–324. https://doi.org/10.1109/ICSEA.2010.55

Alonso, F., Fuertes, J. L., Martínez, L., & Soza, H. (2011). Measures for Evaluating the Software Agent Pro-Activity. In E. Gelenbe, R. Lent, G. Sakellari, A. Sacan, H. Toroslu, & A. Yazici (Eds.), *Computer and Information Sciences* (pp. 61–64). Springer Netherlands. https://doi.org/10.1007/978-90-481-9794-1_12

Antsaklis, P. (2020). Autonomy and metrics of autonomy. *Annual Reviews in Control*, *49*, 15–26. https://doi.org/10.1016/j.arcontrol.2020.05.001

Arora, S., & Sasikala, P. (2016). Quantify autonomy for agent based systems. *International Journal of Control Theory and Applications*, *9*(22), Article 22.

Badri, L., Badri, M., & Toure, F. (2010). Exploring empirically the relationship between lack of cohesion and testability in object-oriented systems. *International Conference on Advanced Software Engineering and Its Applications*, 78–92.

Badri, L., Badri, M., & Toure, F. (2011). An empirical analysis of lack of cohesion metrics for predicting testability of classes. *International Journal of Software Engineering and Its Applications*, *5*(2), 69–85.

Badri, M., Badri, L., Flageol, W., & Toure, F. (2017). Investigating the Accuracy of Test Code Size Prediction using Use Case Metrics and Machine Learning Algorithms: An Empirical Study. *Proceedings of the 2017 International Conference on Machine Learning and Soft Computing*, 25–33. https://doi.org/10.1145/3036290.3036323

Badri, M., Badri, L., Hachemane, O., & Ouellet, A. (2019). Measuring the effect of clone refactoring on the size of unit test cases in object-oriented software: An empirical study. *Innovations in Systems and Software Engineering*, *15*(2), 117–137. https://doi.org/10.1007/s11334-019-00334-6

Badri, M., & Toure, F. (2012). Evaluating the effect of control flow on the unit testing effort of classes: An empirical analysis. *Advances in Software Engineering*, *2012*.

Bagić Babac, M., & Jevtić, D. (2014). AgentTest: A specification language for agent-based system testing. *Neurocomputing*, *146*, 230–248. https://doi.org/10.1016/j.neucom.2014.04.060

Bajeh, A. O., Oluwatosin, O.-J., Basri, S., Akintola, A. G., & Balogun, A. O. (2020). Object-oriented measures as testability indicators: An empirical study. *J. Eng. Sci. Technol*, *15*(2), 1092–1108.

Bakar, N. A., & Selamat, A. (2018). Agent systems verification: Systematic literature review and mapping. *Applied Intelligence*, *48*(5), 1251–1274. https://doi.org/10.1007/s10489-017-1112-z

Barber, K. S., & Martin, C. E. (1999). Agent Autonomy: Specification, Measurement, and Dynamic Adjustment. *Proceedings of the Autonomy Control Software Workshop at Autonomous Agents*, 8–15.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. *Encyclopedia of Software Engineering*, 528–532.

Batista, G. E., Bazzan, A. L., & Monard, M. C. (2003). Balancing training data for automated annotation of keywords: A case study. *Wob*, *3*, 10–18.

Baudet, A., Aktouf, O.-E.-K., Mercier, A., & Jamont, J.-P. (2019). Toward Testing Self-organizations in Multi-Embedded-Agent Systems. In R. Calinescu & F. Di Giandomenico (Eds.), *Software Engineering for Resilient Systems* (pp. 97–108). Springer International Publishing. https://doi.org/10.1007/978-3-030-30856-8_7

Baudry, B., & Le Traon, Y. (2005). Measuring design testability of a UML class diagram. *Information and Software Technology*, *47*(13), 859–879.

Baudry, B., Traon, Y. L., Sunyé, G., & Jézéquel, J.-M. (2004). Measuring and improving design patterns testability. *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717)*, 50–59. https://ieeexplore.ieee.org/abstract/document/1232455/

Belete, D. M., & Huchaiah, M. D. (2022). Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results. *International Journal of Computers and Applications*, *44*(9), 875–886. https://doi.org/10.1080/1206212X.2021.1974663

Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE* (Vol. 7). John Wiley & Sons.

Benaboud, R., & Marir, T. (2020). Flexibility measurement model of multi-agent systems. *Multiagent and Grid Systems*, *16*(3), 309–341. https://doi.org/10.3233/MGS-200334

Berander, P., & Jönsson, P. (2006). Hierarchical cumulative voting (hcv)—Prioritization of requirements in hierarchies. *International Journal of Software Engineering and Knowledge Engineering*, *16*(06), 819–849. https://doi.org/10.1142/S0218194006003026

Bergenti, F., Caire, G., Monica, S., & Poggi, A. (2020). The first twenty years of agent-based software development with JADE. *Autonomous Agents and Multi-Agent Systems*, *34*(2), Article 2. https://doi.org/10.1007/s10458-020-09460-z

Bernstein, B. A., Geurtz, J. C. M., & Koeman, V. J. (2019). Evaluating the Effectiveness of Multi-Agent Organisational Paradigms in a Real-Time Strategy Environment: Engineering Multiagent Systems Track. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 754–762.

Bharat Kumar, T., Harish, N., & Sravan Kumar, V. (2012). An Catholic and Enhanced Study on Basis Path Testing to Avoid Infeasible Paths in CFG. In P. V. Krishna, M. R. Babu, & E. Ariwa (Eds.), *Global Trends in Information Systems and Software Applications* (Vol. 270, pp. 386–395). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29216-3_42

Binder, R. V. (1994). Design for testability in object-oriented systems. *Communications of the ACM*, *37*(9), 87–102.

Boucher, A., & Badri, M. (2018). Software metrics thresholds calculation techniques to predict fault-proneness: An empirical comparison. *Inf. Softw. Technol.*, *96*, 38–67. https://doi.org/10.1016/J.INFSOF.2017.11.005

Braga, P. L., Oliveira, A. L., & Meira, S. R. (2007). Software effort estimation using machine learning techniques with robust confidence intervals. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, 352–357. https://ieeexplore.ieee.org/abstract/document/4344078/

Braynov, S., & Hexmoor, H. (2003). Quantifying Relative Autonomy in Multiagent Interaction. In H. Hexmoor, C. Castelfranchi, & R. Falcone (Eds.), *Agent Autonomy* (pp. 55–73). Springer US. https://doi.org/10.1007/978-1-4419-9198-0_4

Briand, L. C., Morasca, S., & Basili, V. R. (2002). An operational process for goal-driven definition of measures. *IEEE Transactions on Software Engineering*, *28*(12), 1106–1125. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2002.1158285

Brockhoff, K. K. L., & Schmaul, B. (1996). Organization, autonomy, and success of internationally dispersed R D facilities. *IEEE Transactions on Engineering Management*, *43*(1), 33–40. https://doi.org/10.1109/17.491266

Bruntink, M., & Deursen, A. van. (2006). An empirical study into class testability. *Journal of Systems and Software*, *79*(9), 1219–1232. https://doi.org/10.1016/j.jss.2006.02.036

Caire, G. (2009). *JADE Programming-Tutorial-for-beginners*. TILAB, formerly CSELT. https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

Caire, G., Cossentino, M., Negri, A., Poggi, A., & Turci, P. (2004). *Multi-Agent Systems Implementation and Testing*. 11.

Calabrese, J., Esponda, S., Pasini, A., & Pesado, P. (2021). Data Evaluation Model Using GQM Approach. In P. Pesado & J. Eterovic (Eds.), *Computer Science – CACIC 2020* (pp. 141–154). Springer International Publishing. https://doi.org/10.1007/978-3-030-75836-3_10

Canco, I., Kruja, D., & Iancu, T. (2021). AHP, a reliable method for quality decision making: A case study in business. *Sustainability*, *13*(24), 13932.

Carrera, Á., Iglesias, C. A., & Garijo, M. (2014). Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *Information Systems Frontiers*, *16*(2), 169–182. https://doi.org/10.1007/s10796-013-9438-5

Chekam, T. T., Papadakis, M., Le Traon, Y., & Harman, M. (2017). An Empirical Study on Mutation, Statement and Branch Coverage Fault Revelation That Avoids the Unreliable Clean Program Assumption. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 597–608. https://doi.org/10.1109/ICSE.2017.61

Chella, A., Cossentino, M., Sabatucci, L., & Seidita, V. (2004). From passi to agile passi: Tailoring a design process to meet new needs. *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.(IAT 2004).*, 471–474. https://doi.org/10.1109/IAT.2004.1342998

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20*(6), 476–493. https://doi.org/10.1109/32.295895

Coelho, R., Cirilo, E., Kulesza, U., von Staa, A., Rashid, A., & Lucena, C. (2007). JAT: A Test Automation Framework for Multi-Agent Systems. *2007 IEEE International Conference on Software Maintenance*, 425–434. https://doi.org/10.1109/ICSM.2007.4362655

Coelho, R., Kulesza, U., von Staa, A., & Lucena, C. (2006). Unit testing in multi-agent systems using mock agents and aspects. *Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems - SELMAS '06*, 83. https://doi.org/10.1145/1138063.1138079

Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. Academic press. https://books.google.com/books?hl=en&lr=&id=rEe0BQAAQBAJ&oi=fnd&pg=PP1&dq=correlation+weak+(%E2%89%A4+0.3),+moderate+(0.3+%E2%88%92+0.5),+strong++Cohen+J.+Statistical+Power+Analysis+for+the+Behavioral+Sciences.+2nd+ed.1988%3BHillsdale,+NJ:+Lawrence+Erlbaum+Associates,+77%E2%80%9388.&ots=sxTUJpTTuc&sig=50KzZPHbHgHmwCX4_L-Nt8U7boU

Cortese, E., Caire, G., & Bochicchio, R. (2005). *JADE Test Suite User Guide*. TILab. https://jade.tilab.com/doc/tutorials/JADE_TestSuite.pdf

Cossentino, M., Hilaire, V., Molesini, A., & Seidita, V. (Eds.). (2014). *Handbook on Agent-Oriented Design Processes*. Springer-Verlag. https://doi.org/10.1007/978-3-642-39975-6

Cossentino, M., Sabatucci, L., & Chella, A. (2003). Designing JADE systems with the support of CASE tools and patterns. *Special Issue on JADE of Telecom Italia Journal EXP of September*.

Cunha, F., Diniz Da Costa, A., Viana, M., & Pereira De Lucena, C. J. (2015). JAT4BDI: An Aspect-Based Approach for Testing BDI Agents. *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, *2*, 186–189. https://doi.org/10.1109/WI-IAT.2015.121

D. A. Ostrowski & R. G. Reynolds. (1999). Knowledge-based software testing agent using evolutionary learning with cultural algorithms. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, *3*, 1657-1663 Vol. 3. https://doi.org/10.1109/CEC.1999.785473

Dalton, J. (2019). Goal, Question, Metric (GQM). In J. Dalton (Ed.), *Great Big Agile: An OS for Agile Leaders* (pp. 177–179). Apress. https://doi.org/10.1007/978-1-4842-4206-3_33

Dam, H. K., Zhang, T., & Ghose, A. (2013). Improving the reactivity of BDI agent programs. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8291 LNAI*, 85–100. https://doi.org/10.1007/978-3-642-44927-7_7

Darweesh, S. A., Ebrahim, G. A., & Bedour, H. M. S. (2019). Evaluating Multi-Agent System Security using Goal/Question/Metric Approach and Fuzzy Logic. *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 1–6. https://doi.org/10.1109/PACRIM47961.2019.8985125

Deissenboeck, F., Juergens, E., Lochmann, K., & Wagner, S. (2009). Software quality models: Purposes, usage scenarios and requirements. *2009 ICSE Workshop on Software Quality*, 9–14. https://doi.org/10.1109/WOSQ.2009.5071551

Dekhtyar, M., Dikovsky, A., & Valiev, M. (2002). Complexity of Multi-agent Systems Behavior. *European Workshop on Logics in Artificial Intelligence*, 125–136. https://doi.org/10.1007/3-540-45757-7_11

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, *11*(4), 34–41. https://doi.org/10.1109/C-M.1978.218136

Di Bitonto, P., Laterza, M., Roselli, T., & Rossano, V. (2010). An Evaluation Method for Multi-Agent Systems. In P. Jędrzejowicz, N. T. Nguyen, R. J. Howlet, & L. C. Jain (Eds.), *Agent and Multi-Agent Systems: Technologies and Applications* (Vol. 6070, pp. 32–41). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-13480-7_5

Di Bitonto, P., Laterza, M., Roselli, T., & Rossano, V. (2012). Evaluation of multi-agent systems: Proposal and validation of a metric plan. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7270 LNCS*, 198–221. https://doi.org/10.1007/978-3-642-32066-8-9

Dix, J., Hindriks, K., Logan, B., & Wobcke, W. (2012). Engineering Multi-Agent Systems (Dagstuhl Seminar 12342). *Dagstuhl Reports*, *2*(8), 74–98. https://doi.org/10.4230/DagRep.2.8.74

Dix, J., Logan, B., & Winikoff, M. (2021). Preface to the Special Issue on engineering reliable multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, *35*(2), 37. https://doi.org/10.1007/s10458-021-09520-y

Dumke, R., Koeppe, R., & Wille, C. (2000). *Software Agent Measurement and Self Measuring Agent Based Systems*. Univ., Fak. für Informatik.

Dumke, R., Mencke, S., & Wille, C. (2009). *Quality assurance of agent-based and self-managed systems*. CRC Press.

Dziubiński, M., & Verbrugge, R. (2007). Complexity issues in multiagent logics. *Fundamenta Informaticae*, *75*(1–4), 239–262.

Eck, M., Palomba, F., Castelluccio, M., & Bacchelli, A. (2019). Understanding Flaky Tests: The Developer's Perspective. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 830–840. https://doi.org/10.1145/3338906.3338945

Efatmaneshnik, M., & Ryan, M. (2017). A STUDY OF THE RELATIONSHIP BETWEEN SYSTEM TESTABILITY AND MODULARITY. *INSIGHT*, *20*(1), 20–24. https://doi.org/10.1002/inst.12140

Efatmaneshnik, M., Ryan, M. J., & Shoval, S. (2018). A Framework for Testability Analysis from a Systems Architecture Perspective. *INSIGHT*, *21*(3), 72–79. https://doi.org/10.1002/inst.12214

Far, B. H., & Wanyama, T. (2003). Metrics for agent-based software development. *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, *2*, 1297–1300. https://doi.org/10.1109/CCECE.2003.1226137

Ferrando, A., & Malvone, V. (2022). Towards the Combination of Model Checking and Runtime Verification on Multi-agent Systems. In F. Dignum, P. Mathieu, J. M. Corchado, & F. De La

Prieta (Eds.), *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection* (Vol. 13616, pp. 140–152). Springer International Publishing. https://doi.org/10.1007/978-3-031-18192-4_12

Frankl, P. G., Weiss, S. N., & Hu, C. (1997). All-uses vs mutation testing: An experimental comparison of effectiveness. *Journal of Systems and Software*, *38*(3), 235–253. https://doi.org/10.1016/S0164-1212(96)00154-9

Franklin, D., & Abrao, A. (2000). *Measuring Software Agent's Intelligence*. 5.

Franklin, S., & Graesser, A. (1997). Is It an agent, or just a program?: A taxonomy for autonomous agents. In J. P. Müller, M. J. Wooldridge, & N. R. Jennings (Eds.), *Intelligent Agents III Agent Theories, Architectures, and Languages* (Vol. 1193, pp. 21–35). Springer Berlin Heidelberg. https://doi.org/10.1007/BFb0013570

Garousi, V., Felderer, M., & Kılıçaslan, F. N. (2019). A survey on software testability. *Information and Software Technology*, *108*, 35–64. https://doi.org/10.1016/j.infsof.2018.12.003

Gencel, C., Petersen, K., Mughal, A. A., & Iqbal, M. I. (2013). A decision support framework for metrics selection in goal-based measurement programs: GQM-DSFMS. *Journal of Systems and Software*, *86*(12), 3091–3108. https://doi.org/10.1016/j.jss.2013.07.022

Gómez-Sanz, J. J., Botía, J., Serrano, E., & Pavón, J. (2009). Testing and Debugging of MAS Interactions with INGENIAS. In M. Luck & J. J. Gomez-Sanz (Eds.), *Agent-Oriented Software Engineering IX* (pp. 199–212). Springer. https://doi.org/10.1007/978-3-642-01338-6_15

Gonçalves, E. M., Machado, R. A., Rodrigues, B. C., & Adamatti, D. (2022). CPN4M: Testing Multi-Agent Systems under Organizational Model Moise+ Using Colored Petri Nets. *Applied Sciences*, *12*(12), Article 12. https://doi.org/10.3390/app12125857

Gonçalves, E. M., Rodrigues, B. C., & Machado, R. A. (2019). Assessment of Testability on Multiagent Systems Developed with Organizational Model $$\mathcal {M}$$oise. In P. Moura Oliveira, P. Novais, & L. P. Reis (Eds.), *Progress in Artificial Intelligence* (pp. 581–592). Springer International Publishing. https://doi.org/10.1007/978-3-030-30244-3_48

Gutiérrez, C., & García-Magariño, I. (2009). A metrics suite for the communication of multi-agent systems. *Journal of Physical Agents*, *3*(2), Article 2. https://doi.org/10.14198/JoPha.2009.3.2.03

Hamada, D., & Sugawara, T. (2013). Autonomous decision on team roles for efficient team formation by parameter learning and its evaluation. *Intelligent Decision Technologies*, *7*(3), Article 3. https://doi.org/10.3233/IDT-130160

Hannoun, M., Boissier, O., Sichman, J. S., & Sayettat, C. (2000). MOISE: An Organizational Model for Multi-agent Systems. In M. C. Monard & J. S. Sichman (Eds.), *Advances in Artificial Intelligence* (pp. 156–165). Springer. https://doi.org/10.1007/3-540-44399-1_17

Hasebrook, J., Erasmus, L., & Doeben-Henisch, G. (2002). Knowledge Robots for Knowledge Workers: Self-Learning Agents Connecting Information and Skills. In L. C. Jain, Z. Chen, & N. Ichalkaranje (Eds.), *Intelligent Agents and Their Applications* (pp. 59–81). Physica-Verlag HD. https://doi.org/10.1007/978-3-7908-1786-7_3

Hayes, J. H., Li, W., Yu, T., Han, X., Hays, M., & Woodson, C. (2015). Measuring Requirement Quality to Predict Testability. *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 1–8. https://doi.org/10.1109/AIRE.2015.7337622

Henry Coles. (2020). *Mutation operators*. PIT Mutation Testing. https://pitest.org/quickstart/mutators/

Henry Coles, Laurent, T., Henard, C., Papadakis, M., & Ventresque, A. (2016). PIT: A practical mutation testing tool for Java (demo). *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 449–452. https://doi.org/10.1145/2931037.2948707

Hernes, M., Korczak, J., Krol, D., Pondel, M., & Becker, J. (2024). Multi-agent platform to support trading decisions in the FOREX market. *Applied Intelligence*, *54*(22), 11690–11708. https://doi.org/10.1007/s10489-024-05770-x

Hexmoor, H., Castelfranchi, C., & Falcone, R. (Eds.). (2003). *Agent Autonomy* (Vol. 7). Springer US. https://doi.org/10.1007/978-1-4419-9198-0

Hrabia, C.-E., Masuch, N., & Albayrak, S. (2015). A Metrics Framework for Quantifying Autonomy in Complex Systems. In J. P. Müller, W. Ketter, G. Kaminka, G. Wagner, & N. Bulling (Eds.), *Multiagent System Technologies* (Vol. 9433, pp. 22–41). Springer International Publishing. https://doi.org/10.1007/978-3-319-27343-3_2

Huang, C. Y., & Nof, S. Y. (2000). Autonomy and viability-measures for agent-based manufacturing systems. *International Journal of Production Research*, *38*(17), 4129–4148. https://doi.org/10.1080/00207540050204975

Huber, M. J. (2007). Agent Autonomy: Social Integrity and Social Independence. *Fourth International Conference on Information Technology (ITNG'07)*, 282–290. https://doi.org/10.1109/ITNG.2007.29

Inozemtseva, L., & Holmes, R. (2014). Coverage is not strongly correlated with test suite effectiveness. *Proceedings of the 36th International Conference on Software Engineering*, 435–445. https://doi.org/10.1145/2568225.2568271

ISO. (2001). *ISO/IEC 9126-1:2001 Software Engineering – Product Quality*. International Organization for Standardization, Geneva, Switzerland.

ISO. (2011). *ISO/IEC 25010:2011, Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models*. ISO - International Organization for Standardization. https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

ISO. (2022). *ISO/IEC/IEEE 29119:2022(en), Software and systems engineering—Software testing—*. https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-1:ed-2:v1:en

Iwata, K., Nakashima, T., Anan, Y., & Ishii, N. (2016). Effort estimation for embedded software development projects by combining machine learning with classification. *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)*, 265–270. https://ieeexplore.ieee.org/abstract/document/7916993/

Izosimov, V., Ingelsson, U., & Wallin, A. (2012). Requirement Decomposition and Testability in Development of Safety-Critical Automotive Components,. In F. Ortmeier & P. Daniel (Eds.), *Computer Safety, Reliability, and Security* (pp. 74–86). Springer. https://doi.org/10.1007/978-3-642-33678-2_7

Jensen, K. (1997). *Coloured Petri Nets Basic Concepts, Analysis Methods and Practical Use.* (Vol. 1–1). Springer Berlin / Heidelberg. http://public.eblib.com/choice/PublicFullRecord.aspx?p=6502752

Joumaa, H., Demazeau, Y., & Vincent, J.-M. (2008). Evaluation of Multi-Agent Systems: The case of Interaction. *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, 1–6. https://doi.org/10.1109/ICTTA.2008.4530303

Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., & Fraser, G. (2014). Are mutants a valid substitute for real faults in software testing? *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 654–665. https://doi.org/10.1145/2635868.2635929

Kalache, A., Badri, M., Mokhati, F., & Babahenini, M. C. (2023). A testing framework for JADE agent-based software. *Multiagent and Grid Systems*, *19*(1), 61–98.

Keeney, R. L., & Raiffa, H. (1993). *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Cambridge University Press; Cambridge Core. https://doi.org/10.1017/CBO9781139174084

Khan, R. A., & Mustafa, K. (2009). Metric based testability model for object oriented design (MTMOOD). *ACM SIGSOFT Software Engineering Notes*, *34*(2), 1–6. https://doi.org/10.1145/1507195.1507204

Klügl, F. (2008). Measuring Complexity of Multi-agent Simulations – An Attempt Using Metrics. In M. Dastani, A. El Fallah Seghrouchni, J. Leite, & P. Torroni (Eds.), *Languages, Methodologies and Development Tools for Multi-Agent Systems* (Vol. 5118, pp. 123–138). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85058-8_8

Kravari, K., & Bassiliades, N. (2015). A Survey of Agent Platforms. *Journal of Artificial Societies and Social Simulation*, *18*(1), 11. https://doi.org/10.18564/jasss.2661

Lam, W., Muşlu, K., Sajnani, H., & Thummalapenta, S. (2020). A study on the lifecycle of flaky tests. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 1471–1482. https://doi.org/10.1145/3377811.3381749

Leffingwell, D., & Widrig, D. (2000). *Managing software requirements: A unified approach*. Addison-Wesley Professional.

Leitão, P., & Karnouskos, S. (2015). *Industrial Agents: Emerging Applications of Software Agents in Industry*. Morgan Kaufmann. https://doi.org/10.1016/C2013-0-15269-5

Leotta, M., Cerioli, M., Olianas, D., & Ricca, F. (2018). Fluent vs Basic Assertions in Java: An Empirical Study. *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 184–192. https://doi.org/10.1109/QUATIC.2018.00036

Leotta, M., Cerioli, M., Olianas, D., & Ricca, F. (2019). Hamcrest vs AssertJ: An Empirical Assessment of Tester Productivity. In M. Piattini, P. Rupino da Cunha, I. García Rodríguez de Guzmán, & R. Pérez-Castillo (Eds.), *Quality of Information and Communications Technology* (pp. 161–176). Springer International Publishing. https://doi.org/10.1007/978-3-030-29238-6_12

Leotta, M., Cerioli, M., Olianas, D., & Ricca, F. (2020). Two experiments for evaluating the impact of Hamcrest and AssertJ on assertion development. *Software Quality Journal*, *28*(3), 1113–1145. https://doi.org/10.1007/s11219-020-09507-0

Low, C. K., Chen, T. Y., & Rónnquist, R. (1999). Automated Test Case Generation for BDI Agents. *Autonomous Agents and Multi-Agent Systems*, *2*(4), 311–332. https://doi.org/10.1023/A:1010011219782

Luo, Q., Hariri, F., Eloussi, L., & Marinov, D. (2014). An empirical analysis of flaky tests. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 643–653. https://doi.org/10.1145/2635868.2635920

M. Mala & İ. Çil. (2011). A taxonomy for measuring complexity in agent-based systems. *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, 851–854. https://doi.org/10.1109/ICSESS.2011.5982474

Machado, R. A., & Gonçalves, E. M. (2020). Testing Multiagent Systems Under Organizational Model $$\mathcal {M}$$oise Using a Test Adequacy Criterion Based on State Transition Path. In R. Cerri & R. C. Prati (Eds.), *Intelligent Systems* (pp. 154–168). Springer International Publishing. https://doi.org/10.1007/978-3-030-61380-8_11

Mahar, S., & Bhatia, P. K. (2014). Measuring the intelligence of software agent. *Int. J. Innovative Sci. Eng. Technol*, *1*(6), 1–11.

Marir, T., Mokhati, F., Bouchelaghem-Seridi, H., & Benaissa, B. (2016). Dynamic Metrics for Multi-agent Systems Using Aspect-Oriented Programming. In M. Klusch, R. Unland, O. Shehory, A. Pokahr, & S. Ahrndt (Eds.), *Multiagent System Technologies* (Vol. 9872, pp. 58–72). Springer International Publishing. https://doi.org/10.1007/978-3-319-45889-2_5

Marir, T., Mokhati, F., Bouchelaghem-Seridi, H., & Tamrabet, Z. (2014). Complexity Measurement of Multi-Agent Systems. In J. P. Müller, M. Weyrich, & A. L. C. Bazzan (Eds.), *Multiagent System Technologies* (Vol. 8732, pp. 188–201). Springer International Publishing. https://doi.org/10.1007/978-3-319-11584-9_13

Marir, T., Mokhati, F., Bouchlaghem-Seridi, H., Acid, Y., & Bouzid, M. (2016). QM4MAS: A Quality Model For Multi-Agent Systems. *International Journal of Computer Applications in Technology*, *54*(4), 297–310. https://doi.org/10.1504/IJCAT.2016.080485

Mascardi, V., Weyns, D., Ricci, A., Earle, C. B., Casals, A., Challenger, M., Chopra, A., Ciortea, A., Dennis, L. A., Díaz, Á. F., El Fallah-Seghrouchni, A., Ferrando, A., Fredlund, L.-Å., Giunchiglia, E., Guessoum, Z., Günay, A., Hindriks, K., Iglesias, C. A., Logan, B., … Winikoff, M. (2019). Engineering Multi-Agent Systems: State of Affairs and the Road Ahead. *ACM SIGSOFT Software Engineering Notes*, *44*(1), 18–28. https://doi.org/10.1145/3310013.3322175

Massimo Cossentino. (2005). From Requirements to Code with PASSI Methodology. In Brian Henderson-Sellers & Paolo Giorgini (Eds.), *Agent-Oriented Methodologies* (pp. 79–106). IGI Global. https://doi.org/10.4018/978-1-59140-581-8.ch004

Massonet, P., Deville, Y., & Nève, C. (2002). From AOSE methodology to agent implementation. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, 27–34. https://doi.org/10.1145/544741.544747

Matcha, W., Touré, F., Badri, M., & Badri, L. (2022). Identifying Candidate Classes for Unit Testing Using Deep Learning Classifiers: An Empirical Validation. *Proceedings of the 4th World Symposium on Software Engineering, WSSE 2022, Xiamen, China, September 28-30, 2022*, 98–107. https://doi.org/10.1145/3568364.3568380

McCabe, T. J. (1976a). A Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), 308–320. https://doi.org/10.1109/TSE.1976.233837

McCabe, T. J. (1976b). A Complexity Measure. *IEEE Transactions on Software Engineering*, *SE-2*(4), Article 4. https://doi.org/10.1109/TSE.1976.233837

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 51–56). https://doi.org/10.25080/Majora-92bf1922-00a

Meixner, A., & Gullo, L. J. (2021). Design for Test and Testability. In L. J. Gullo & J. Dixon (Eds.), *Design for Maintainability* (1st ed., pp. 245–264). Wiley. https://doi.org/10.1002/9781119578536.ch13

Moraïtis, P., Petraki, E., & Spanoudakis, N. I. (2003). Engineering JADE Agents with the Gaia Methodology. In J. G. Carbonell, J. Siekmann, R. Kowalczyk, J. P. Müller, H. Tianfield, & R. Unland (Eds.), *Agent Technologies, Infrastructures, Tools, and Applications for E-Services* (pp. 77–91). Springer. https://doi.org/10.1007/3-540-36559-1_8

Mostafa, S. A., Ahmad, M. S., Ahmad, A., Annamalai, M., & Mustapha, A. (2014). A dynamic measurement of agent autonomy in the layered adjustable autonomy model. *Studies in Computational Intelligence*, *513*, 25–35. https://doi.org/10.1007/978-3-319-01787-7_3

Moudache, S., & Badri, M. (2022). Using Metrics for Risk Prediction in Object-Oriented Software: A Cross-Version Validation. *J. Softw., 17*(1), 1–20. https://doi.org/10.17706/JSW.17.1.1-20

Müller, J. P., & Fischer, K. (2014). Application Impact of Multi-agent Systems and Technologies: A Survey. In O. Shehory & A. Sturm (Eds.), *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks* (pp. 27–53). Springer. https://doi.org/10.1007/978-3-642-54432-3_3

Munroe, S., Miller, T., Belecheanu, R. A., Pěchouček, M., Mcburney, P., & Luck, M. (2006). Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *The Knowledge Engineering Review, 21*(4), 345–392. https://doi.org/10.1017/S0269888906001020

NASA. (2017). *NASA Reliability and Maintainability (R&M) Standard for Spaceflight and Support Systems* (No. NASA-STD-8729.1 A).

Nascimento, N., Viana, C. J., Staa, A., & Lucena, C. (2017). *A Publish-Subscribe based Architecture for Testing Multiagent Systems*. 521–526. https://doi.org/10.18293/SEKE2017-050

Nguyen, C. D., Perini, A., Bernon, C., Pavón, J., & Thangarajah, J. (2009). Testing in Multi-Agent Systems. In M.-P. Gleizes & J. J. Gomez-Sanz (Eds.), *Agent-Oriented Software Engineering X* (pp. 180–190). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-19208-1_13

Nguyen, C. D., Perini, A., & Tonella, P. (2008). A Goal-Oriented Software Testing Methodology. In M. Luck & L. Padgham (Eds.), *Agent-Oriented Software Engineering VIII: 8th International Workshop, AOSE 2007, Honolulu, HI, USA, May 14, 2007, Revised Selected Papers* (pp. 58–72). Springer. https://doi.org/10.1007/978-3-540-79488-2_5

Nguyen, C. D., Perini, A., & Tonella, P. (2010). Goal-oriented testing for MASs. *International Journal of Agent-Oriented Software Engineering, 4*(1), Article 1. https://doi.org/10.1504/IJAOSE.2010.029810

North, D. (2006). Introducing BDD. *Dan North & Associates*. https://dannorth.net/introducing-bdd/

Ouellet, A., & Badri, M. (2019). Empirical Analysis of Object-Oriented Metrics and Centrality Measures for Predicting Fault-Prone Classes in Object-Oriented Software. In M. Piattini, P. R. da Cunha, I. G. R. de Guzmán, & R. Pérez-Castillo (Eds.), *Quality of Information and Communications Technology—12th International Conference, QUATIC 2019, Ciudad Real, Spain, September 11-13, 2019, Proceedings* (Vol. 1010, pp. 129–143). Springer. https://doi.org/10.1007/978-3-030-29238-6_10

Ouellet, A., & Badri, M. (2024). Combining object-oriented metrics and centrality measures to predict faults in object-oriented software: An empirical validation. *J. Softw. Evol. Process.*, *36*(4). https://doi.org/10.1002/SMR.2548

Pant, S., Kumar, A., Ram, M., Klochkov, Y., & Sharma, H. K. (2022). Consistency indices in analytic hierarchy process: A review. *Mathematics*, *10*(8), 1206.

Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. (2021). Benchmarking Multi-Agent Deep Reinforcement Learning Algorithms in Cooperative Tasks. *arXiv:2006.07869 [Cs, Stat]*. http://arxiv.org/abs/2006.07869

Payne, J. E., Alexander, R. T., & Hutchinson, C. D. (1997). Design-for-testability for object-oriented software. *Object Magazine*, *7*(5), 34–43.

Petrillo, A., Salvi, A., Santini, S., & Valente, A. S. (2018). Adaptive multi-agents synchronization for collaborative driving of autonomous vehicles with multiple communication delays. *Transportation Research Part C: Emerging Technologies*, *86*, 372–392.

Quintero-Parra, A. F., Camacho-Navarro, J., Flórez, M., & Vázquez-González, J. L. (2017). Evaluation of multi-agent architecture for structural damage detection and location. *International Journal of Online Engineering*, *13*(6), Article 6. https://doi.org/10.3991/ijoe.v13i06.7184

Ramirez, C. A., Thompson, A. E., & Gorthala, R. (2024). A Design for Testability (DFT) strategy for the development of highly complex safety-critical system using a Model-Based Systems Engineering (MBSE) approach. *2024 IEEE International Symposium on Systems Engineering (ISSE)*, 1–8. https://ieeexplore.ieee.org/abstract/document/10741151/

Rouhani, S., & Mirhosseini, S. V. (2015). Development and evaluation of intelligent agent- based teaching assistant in e-learning portals. *International Journal of Web-Based Learning and Teaching Technologies*, *10*(4), Article 4. https://doi.org/10.4018/IJWLTT.2015100104

S. Chichin, M. B. Chhetri, Q. B. Vo, R. Kowalczyk, & M. Stepniak. (2014). Smart Cloud Marketplace—Agent-Based Platform for Trading Cloud Services. *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, *3*, 388–395. https://doi.org/10.1109/WI-IAT.2014.193

Saaty, T. L. (1988). What is the Analytic Hierarchy Process? In G. Mitra, H. J. Greenberg, F. A. Lootsma, M. J. Rijkaert, & H. J. Zimmermann (Eds.), *Mathematical Models for Decision Support* (pp. 109–121). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-83555-1_5

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, *48*(1), Article 1. https://doi.org/10.1016/0377-2217(90)90057-I

Sanislav, T., Zeadally, S., Mois, G. D., & Fouchal, H. (2018). Reliability, failure detection and prevention in cyber-physical systems (CPSs) with agents. *Concurrency Computation*. https://doi.org/10.1002/cpe.4481

Sarkar, A., & Debnath, N. C. (2012). Measuring complexity of Multi-Agent System architecture. *IEEE 10th International Conference on Industrial Informatics*, 998–1003. https://doi.org/10.1109/INDIN.2012.6300838

Satapathy, S. M., Acharya, B. P., & Rath, S. K. (2016). Early stage software effort estimation using random forest technique based on use case points. *IET Software*, *10*(1), 10–17. https://doi.org/10.1049/iet-sen.2014.0122

Sayward, F., DeMillo, R., Budd, T. A., & Lipton, R. J. (1978). The design of a prototype mutation system for program testing. *Managing Requirements Knowledge, International Workshop On*, 623. https://doi.org/10.1109/AFIPS.1978.195

Shobole, A. A., & Wadi, M. (2021). Multiagent systems application for the smart grid protection. *Renewable and Sustainable Energy Reviews*, *149*, 111352.

Siraj, S., Mikhailov, L., & Keane, J. A. (2015). PriEsT: An interactive decision support tool to estimate priorities from pairwise comparison judgments: PriEsT: an interactive decision support tool to estimate priorities from pairwise comparison judgments. *International Transactions in Operational Research*, *22*(2), 217–235. https://doi.org/10.1111/itor.12054

Sivakumar, N., & Vivekanandan, K. (2012). Measures for Testing the Reactivity Property of a Software Agent. *International Journal of Advanced Research in Artificial Intelligence*, *1*(9). https://doi.org/10.14569/IJARAI.2012.010905

Sivakumar, N., Vivekanandan, K., & Sandhya, S. (2011). Testing Agent-Oriented Software by Measuring Agent's Property Attributes. In A. Abraham, J. Lloret Mauri, J. F. Buford, J. Suzuki, & S. M. Thampi (Eds.), *Advances in Computing and Communications* (Vol. 191, pp. 88–98). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-22714-1_10

Solingen, R. van, & Berghout, E. (1999). *The Goal/Question/Metric Method: A practical guide for quality improvement of software development*. The McGraw-Hill Companies.

Soza, H. (2018). Quality Measures for Agent-Oriented Software. In V. Shikhin (Ed.), *Multi-Agent Systems—Control Spectrum*. IntechOpen. https://doi.org/10.5772/intechopen.79741

Sturm, A., & Shehory, O. (2014a). Agent-Oriented Software Engineering: Revisiting the State of the Art. In O. Shehory & A. Sturm (Eds.), *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks* (pp. 13–26). Springer. https://doi.org/10.1007/978-3-642-54432-3_2

Sturm, A., & Shehory, O. (2014b). The Landscape of Agent-Oriented Methodologies. In O. Shehory & A. Sturm (Eds.), *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks* (pp. 137–154). Springer. https://doi.org/10.1007/978-3-642-54432-3_7

Terragni, V., Salza, P., & Pezzè, M. (2020). Measuring Software Testability Modulo Test Quality. *Proceedings of the 28th International Conference on Program Comprehension*, 241–251. https://doi.org/10.1145/3387904.3389273

Tiryaki, A. M., Öztuna, S., Dikenelli, O., & Erdur, R. C. (2007). SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems. In L. Padgham & F. Zambonelli (Eds.), *Agent-Oriented Software Engineering VII* (pp. 156–173). Springer. https://doi.org/10.1007/978-3-540-70945-9_10

Toure, F., Badri, M., & Lamontagne, L. (2014). A metrics suite for JUnit test code: A multiple case study on open source software. *Journal of Software Engineering Research and Development*, *2*(1), 14. https://doi.org/10.1186/s40411-014-0014-6

Touré, F., Badri, M., & Lamontagne, L. (2018). Predicting different levels of the unit testing effort of classes using source code metrics: A multiple case study on open-source software. *Innov. Syst. Softw. Eng.*, *14*(1), 15–46. https://doi.org/10.1007/S11334-017-0306-1

Vahi, Y., & Ignise, A. (2024). Designing A Multi-Agent System For Smart Grid Management. *Authorea Preprints*. https://www.techrxiv.org/doi/full/10.36227/techrxiv.171838939.98209610

Van Liedekerke, M. H., & Avouris, N. M. (1995). Debugging multi-agent systems. *Information and Software Technology*, *37*(2), 103–112. https://doi.org/10.1016/0950-5849(95)93487-Y

Vig, V., & Kaur, A. (2018). Test effort estimation and prediction of traditional and rapid release models using machine learning algorithms. *Journal of Intelligent & Fuzzy Systems*, *35*(2), 1657–1669. https://doi.org/10.3233/JIFS-169703

Voas, J. M. (1992). PIE: A dynamic failure-based technique. *IEEE Transactions on Software Engineering*, *18*(8), 717–727. IEEE Transactions on Software Engineering. https://doi.org/10.1109/32.153381

Wang, X. (2024). *Multi-agent systems for smart grids: Revolutionizing energy management*. https://lutpub.lut.fi/handle/10024/167152

Watson, A. H., Wallace, D. R., & McCabe, T. J. (1996). *Structured testing: A testing methodology using the cyclomatic complexity metric* (Vol. 500). US Department of Commerce, Technology Administration, National Institute of ….

Wille, C., Brehmer, N., & Dumke, R. R. (2004). *Software measurement of agent-based systems an evaluation study of the agent academy* (p. 87). Univ., Fak. für Informatik.

Wille, C., Dumke, R., & Stojanov, S. (2002). *Quality Assurance in Agent Based Systems: Current State and Open Problems*. Otto-von-Guericke University of Magdeburg.

Winikoff, M. (2012). Challenges and Directions for Engineering Multi-agent Systems. *Engineering Multi-Agent Systems*. Dagstuhl Seminar 12342 (August 2012), Dagstuhl. http://arxiv.org/abs/1209.1428

Winikoff, M. (2017). BDI agent testability revisited. *Autonomous Agents and Multi-Agent Systems*, *31*(5), Article 5. https://doi.org/10.1007/s10458-016-9356-2

Winikoff, M., & Cranefield, S. (2014). On the Testability of BDI Agent Systems. *Journal of Artificial Intelligence Research*, *51*, 71–131. https://doi.org/10.1613/jair.4458

Winikoff, M., & Cranefield, S. (2015). On the Testability of BDI Agent Systems (Extended Abstract). *Proceedings of the International Joint Conference on Artificial Intelligence*, 4217–4221. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI15/paper/view/10717

Winikoff, M., & Padgham, L. (2013). Agent oriented software engineering. In *Multiagent systems* (2nd ed., pp. 695–757). MIT Press.

Wooldridge, M. (2001). *Introduction to Multiagent Systems*. John Wiley & Sons, Inc.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *10*(2), 115–152. https://doi.org/10.1017/S0269888900008122

Wrona, Z., Buchwald, W., Ganzha, M., Paprzycki, M., Leon, F., Noor, N., & Pal, C.-V. (2023). Overview of software agent platforms available in 2023. *Information*, *14*(6), 348.

Xenos, M., Stavrinoudis, D., Zikouli, K., & Christodoulakis, D. (2000). Object-oriented metrics-a survey. *Proceedings of the FESMA*, 1–10.

Yahya, F., Walters, R. J., & Wills, G. B. (2017). Using Goal-Question-Metric (GQM) Approach to Assess Security in Cloud Storage. In V. Chang, M. Ramachandran, R. J. Walters, & G. Wills (Eds.), *Enterprise Security* (pp. 223–240). Springer International Publishing. https://doi.org/10.1007/978-3-319-54380-2_10

Zakeri-Nasrabadi, M., & Parsa, S. (2024). Natural language requirements testability measurement based on requirement smells. *Neural Computing and Applications*, *36*(21), 13051–13085. https://doi.org/10.1007/s00521-024-09730-x

Zakeri-Nasrabadi, M., Parsa, S., & Jafari, S. (2024). Measuring and improving software testability at the design level. *Information and Software Technology*, *174*, 107511.

Zambonelli, F., & Omicini, A. (2004). Challenges and Research Directions in Agent-Oriented Software Engineering. *Autonomous Agents and Multi-Agent Systems*, *9*(3), Article 3. https://doi.org/10.1023/B:AGNT.0000038028.66672.1e

Zuparic, M., Jauregui, V., Prokopenko, M., & Yue, Y. (2017). Quantifying the impact of communication on performance in multi-agent teams. *Artificial Life and Robotics*, *22*(3), Article 3. https://doi.org/10.1007/s10015-017-0367-0