2005

Comparing Agent Software Development Methodologies using the CMMI Engineering Process Model

Sarah Bassett Lee
University of Memphis, sblee@memphis.edu

Sajjan Shiva
University of Memphis, sshiva@memphis.edu

Comparing Agent Software Development Methodologies using the CMMI Engineering Process Model

Sarah Bassett Lee
University of Memphis
sblee@memphis.edu

Sajjan Shiva
University of Memphis
sshiva@memphis.edu

ABSTRACT

This paper explores the fulfillment of Capability Maturity Model Integration (CMMI) processes and practices by two widely published agent-based software development methodologies: Multiagent Systems Engineering Methodology (MaSE) and Gaia. Comparisons of these methodologies have been documented and published, but no known research exists that compares and contrasts these methodologies with the CMMI processes.

CMMI provides a benchmark for mature, capable processes, identifying practices necessary to implement those processes effectively. In this paper, the engineering process areas of CMMI are used as a baseline for a theoretical analysis of capabilities offered by software development methodologies designed specifically for the agent-based software domain. An overview of the engineering process areas along with general descriptions of MaSE and Gaia are included. Differences between these methodologies are explored in a functional mapping to the engineering process areas of CMMI. The analysis will show that, while each methodology examined offers guidance in the design and development of solutions, guidance for implementation is minimal. Additionally, verification and validation techniques, critical to effective software engineering, are not addressed sufficiently by MaSE or Gaia.

Keywords

CMMI, Agent Methodology, MaSE, Gaia.

INTRODUCTION

An agent is a computer system that exists in an environment in which it is capable of responsive and proactive actions in order to meet its objectives. An agent may respond to changes in its environment or may exhibit opportunistic behavior and take initiative when appropriate (Knublauch, 2002). The complex interactions among agents or between agents and other aspects of their environment are what differentiate them from other forms of software.

In a multi-agent system, where agents are designed and implemented to interact with one another, these distinct yet cooperative software systems present an opportunity for a non-traditional approach to software engineering. These multiple encapsulated systems may be approached with different designs and diverse development resources (Dam and Winikoff, 2003). Development processes must enable the creation of these flexible, yet complex agent-based software systems and be able to support intra-agent and inter-agent characteristics. With agent implementations gravitating towards the incorporation of social and cognitive concepts in the design, implementation specifics such as operational environment must also be considered in the development process (Dastani, Hulstijn, Dignum, and Meyer, 2004).

Methodologies supporting agent-based development have proliferated over recent years. Literature review reveals substantial analysis of MaSE and Gaia, with implementations primarily in academia. Comparisons of these methodologies have been documented and published, but no known research exists that compares and contrasts these methodologies with the Capability Maturity Model Integration (CMMI) processes.

CMMI provides a baseline for integrating practices and procedures supporting both software and systems engineering. Through CMMI, a benchmark is provided for mature, capable processes, identifying practices necessary to implement those processes effectively. In this paper, CMMI is used a baseline for comparing and contrasting the capabilities offered by software development methodologies designed specifically for the agent-based software domain.

AGENT-BASED METHODOLOGIES

The creation of this inherently complex agent-based software must be supported by adequate processes (Wood and DeLoach, 2000). The primary goal of the MaSE methodology is depicted as providing a complete lifecycle methodology for design and development of multi-agent systems (Sudeikat, Braubach, Pokahr, and Lamersdorf, 2004). Gaia is described as providing at most a level of abstract design for a multi-agent system (Wooldridge, Jennings, and Kinny, 2000). These methodologies will be examined in terms of CMMI Engineering process requirements to determine their differences and the extent of their guidance for software design and development.

Multi-Agent Software Engineering

The MaSE methodology consists of two main phases: Analysis and Design. Tool support is provided throughout the phases with agentTool, which is intended to transform analysis models into design constructs. Practices and deliverables in the MaSE analysis and design phases are depicted in Figure 1 (Wood and DeLoach, 2000).

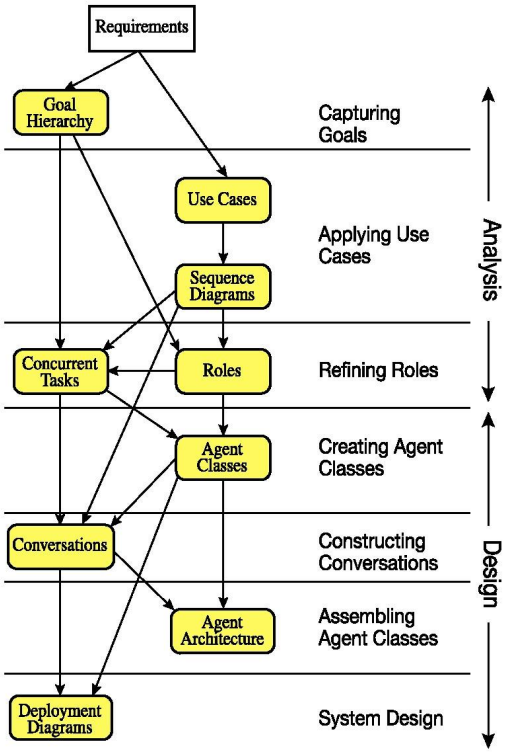

Figure 1. MaSE Phases (Wood and DeLoach, 2000)

The Analysis stage consists of three steps: Capturing Goals, Applying Use Cases, and Refining Roles. In the Capturing Goals step, the goals are identified and represented in a *Goal Hierarchy Diagram*. When Applying Use Cases is conducted, the main scenarios are extracted from the initial system context. These scenarios are used to build a set of *Sequence Diagrams*. The final step in the Analysis stage is Refining Roles, in which a *Role Model* and a C*oncurrent Task Model* are constructed. The Role Model describes the roles in the system and the goals for which each role is responsible. Also depicted are the tasks that each role performs and the communication paths between roles. The tasks are represented at a lower level of detail in the *Concurrent Task Model*.

In the Design stage, Agent Classes are created and depicted in an *Agent Class Diagram* showing agents and the roles they play. Conversations between agents are also depicted in the Agent Class Diagram. Details of the conversations are described in a *Communication Class Diagram*, the output of the Constructing Conversations step of the Design stage. Next, the agent architecture is defined down to the component level. However, no implementation platform is specified. The final step of this methodology is the System Design step where the locations of agents within a system are specified in a *Deployment Diagram* (DeLoach, 2001).

Gaia

Within the Gaia agent-oriented methodology, guidance is provided from requirements statements through analysis and design in a process of developing models that increase in detail (Wooldridge, et al., 2000). These models are summarized in Figure 2 below.

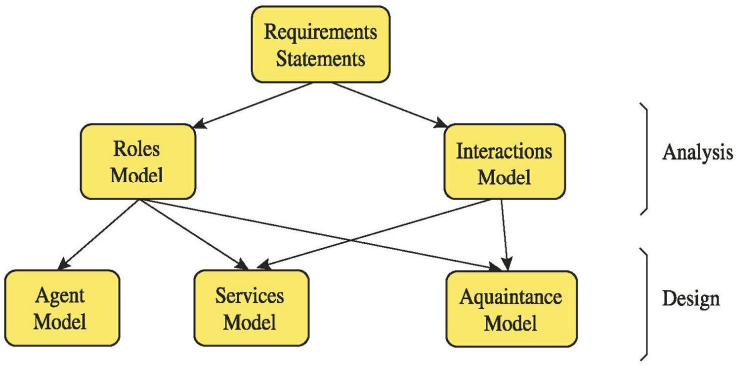

Figure 2. Gaia Phases (Wooldridge, et al., 2000)

In the analysis stage, an understanding of the system is derived through the building of two models, the *Roles Model* and the *Interaction Model*. In the Role Model, the system is represented through its roles, which are defined based upon responsibilities, permissions, activities, and protocols. Responsibilities, which determine functionality, are key to each role. Responsibilities are categorized into *liveness properties* and *safety properties*. Liveness properties state the results of a role's activities within certain environmental conditions. Safety properties are conditions that must hold across all of a role's activities.

The interactions between the roles are depicted in the *Interaction Model*. This model consists of protocol definitions with emphasis on the purpose of each interaction rather than the ordering of message exchanges. The Roles Model is elaborated based on the Interaction Model.

In the Gaia design stage, the models delivered out of analysis are transformed into a lower level of abstraction. However, implementation level details are beyond the scope of the Gaia methodology. The models produced during this stage include the *Agent Model*, the *Services Model*, and the *Acquaintance Model*. The Agent Model documents the agent types and instances that will comprise the system. In the Services Model, the functions associated with each agent are depicted. In the Acquaintance Model, the communication between agents is represented, identifying possible bottlenecks that may occur at run-time (Wooldridge, et al., 2000).

CAPABILITY MATURITY MODEL INTEGRATION

CMMI was introduced in 2002 by Carnegie Mellon's Software Engineering Institute. It is designed as a guide for integration of the variety of practices and procedures found in capability maturity models supporting both software and systems engineering. A capability maturity model delineates the characteristics of a mature, capable process, identifying practices necessary to implement effective processes. CMMI provides an integrated capability model that integrates process improvement guidance for systems engineering and software engineering, reducing the redundancy and complexity of using separate maturity models. CMMI can and should be tailored to an organization's mission and business objectives (CMMI Product Team, 2002).

Following CMMI provides an organization with a framework onto which activities that make up a system development lifecycle can be mapped and tuned so that continuous improvement of those processes becomes a consistent part of the overall lifecycle. The CMMI continuous model enables an organization to optimize their processes in a particular area without pursuing certification for the entire organization. With this model, a particular process is identified and moved through the capability levels to full optimization. This enables an organization to focus on those processes that are immediately critical. CMMI also provides a staged model, which uses maturity levels to measure process improvement (CMMI Product Team, 2002).

The capability levels determine what should be focused on for each process area. Level Zero of CMMI capability represents incomplete processes. At Level One, processes are merely performed. Level Two represents processes that are performed,

planned, tracked, and controlled within a specific group only. At Level Three, processes are performed and managed within the entire organization. At Level Four, an organization is able to extract statistical measurements regarding the success of a process. With Level Five, a process is optimized through continuous process improvement (CMMI Product Team, 2002).

Processes within CMMI are grouped by process areas. Process areas are grouped into four high level categories: Process Management, Project Management, Engineering, and Support. In order to produce a foundation upon which a comparison of agent-based methodologies with CMMI will be made, the Engineering process areas will be described in more detail. This is the CMMI category with objectives most similar to those of software engineering methodologies.

The purpose of each Engineering Process Area is summarized in Table 1:

Process Area	Purpose
Requirements Management	Manage requirements; Identify inconsistencies between requirements and project plans and work products
Requirements Development	Produce and analyze customer, product, and component requirements
Technical Solution	Design, develop, and implement solutions to requirements
Product Integration	Assemble the product from components and ensure that the integrated product functions properly, then deliver the product
Verification	Ensure the selected work products meet their specified requirements
Validation	Demonstrate that a product or component fulfills its intended use when placed in its intended environment

Table 1. Engineering Process Areas

ANALYSIS

For each process area, specific goals and related specific practices will be referenced. Specific goals describe what must be implemented to satisfy the process area. A specific practice is considered important in achieving the corresponding specific goal.

Each specific goal is listed with an 'SG' followed by a number denoting a sequential numbering scheme. Each specific practice is represented with 'SP' followed by a number in the form *x.y-z*. The *x* represents the number of the goal to which the practice is mapped. The *y* is the sequence number of the practice under the specific goal. The *z* represents the capability level of the specific practice. Those practices at capability level 1 are called 'base practices'; all others are 'advanced practices' (CMMI Product Team, 2002).

Examining MaSE

In the first step of the MaSE methodology, Capturing Goals, the user requirements are transformed into top-level system goals. This assumes that the customer requirements have already been developed. MaSE does not support the first specific goal of the Requirements Development process area, *[SG1] Develop Customer Requirements*. There is also no support within the MaSE methodology for requirements management, thus the Requirements Management process area is unfulfilled with this methodology.

When following the MaSE methodology, *[SG2] Develop Product Requirements* is the first specific goal supported. By evaluating the set of user requirements, the system level goals are defined. These goals are structured in a Goal Hierarchy Diagram, which enables the analyst to organize the goals by importance and represent specific goals required for the completion of parent goals. However, it is later in the methodology, when Sequence Diagrams are produced, that interface requirements are identified per CMMI *[SP 2.3-1] Identify Interface Requirements.*

In the Applying Use Cases step, the goals are translated into roles and associated tasks. Based on the system requirements, use cases are produced describing the sequence of events that will define the system behavior. The use cases support the *[SP 1.2-2] Evolve operational concepts and scenarios* specific practice of the Technical Solution Process area.

In the Sequence Diagrams that are produced in MaSE's Analysis stage, the basic communications between agents are defined. In the next Analysis step, Refining Roles, each goal is mapped to a role. When roles are fully defined, high-level tasks are created. This step supports the *[SP 3.2-1] Establish definition of required functionality* subpractice of the Requirements Development process area.

In the Refining Roles step, the high-level tasks for each role are expanded to a lower level of detail in the Concurrent Task Diagram. Series of messages between the tasks are depicted, supporting *[SP 3.3-1] Analyze Requirements* of Requirements Development.

In the first step of the MaSE Design Stage, agent classes are identified based on the previously defined roles. During this step, the analyst may combine roles into a single class or map a single role to multiple classes in order to make the organization more efficient. The analyst identifies the agent classes that will make up the multi-agent system.

The next two steps in the Design stage, Constructing Conversations and Assembling Agent Classes, may be conducted in parallel. A conversation defines a coordination protocol between two agents, and consists of a Communication Class Diagram for the initiator and for the responder. Each agent conversation is compared with other active conversations (DeLoach, 2001). This provides verification of conversation correctness, supporting the Verification process area for agent communications. The conversations and agent architectures that are delivered in these Design steps support *[SP 2.1-1] Design Product or Component* and *[SP 2.3-1] Establish Interface Descriptions* of the Technical Solution process area.

In the System Design step within MaSE, the overall system architecture is defined, showing the numbers, types, and locations of agents within the system. This is the step in which implementation decisions such as programming language are made. This implementation-specific decision making process supports, in a limited fashion, *[SP 2.4-3] Perform Make, Buy, or Reuse Analysis*. While not directly providing guidelines for choices in acquisition, the design phase of MaSE provides the opportunity to evaluate opportunities to re-use existing agents and to evaluate the appropriate implementation technology based on technical requirements and skill set availability.

In the Technical Solution *[SP 2.2-3] Establish technical data package* subpractice, a comprehensive description of the product's architecture is provided and should be maintained throughout the life of the product to record details of the design. The Deployment Diagram provides artifacts in support of this CMMI feature. Given that the Deployment Diagram denotes the number, type, and locations of the agents that will comprise the multi-agent system, this deliverable can also be mapped to *[SP 1.1-1] Determine integration sequence* of the first specific goal of the Product Integration process area (DeLoach, 2001). In support of this specific goal, the product components to be integrated are identified, and verifications are performed on the interfaces between components.

The agentTool product that supports the MaSE methodology provides limited support of *[SG3] Implement the Product Design.* Java code templates can be generated, but not implementable code (DeLoach, 2001). MaSE also does not provide guidelines for producing end-user and support documentation, as described in *[SP 3.2-1] Develop Product Support Documentation.* Given that MaSE's guidance ends with the code templates producible within agentTool, there is no support for the Product Integration process area other than the first specific practice of *[SG1] Prepare for Product Integration.*

The purpose of Verification is to assemble the product from its components and ensure that the product functions properly before delivery. This includes verification of intermediate work products against customer and product requirements (CMMI Product Team, 2002). This process area is embraced by the MaSE methodology for agent communications only. As the Validate process area involves demonstrating that the product fulfills its intended use when placed in its target environment, it is clear that no direct support is provided for this process area within the MaSE methodology (CMMI Product Team, 2002).

Examining Gaia

In the Gaia methodology, the process of gathering requirements is viewed as independent from the stages of analysis and design. Thus, the Requirements Management and Requirements Development process areas are not supported.

In the analysis stage, the evolution of the Roles and Interaction models supports the *[SP 1.1-2] Evolve operational concepts and scenarios* specific practice of the Technical Solution Process area. Responsibilities are categorized using liveness and safety properties, which relate to the operational environment in which the roles will exist.

In the design process of Gaia, support for two specific goals supporting the Technical Solution process area is identified. Component solutions are reflected in the Agent Model, which documents the types and instances of agents that will comprise the multi-agent system. This supports *[SP 1.3-1] Select Component Solutions* of the first specific goal, *[SG1] Select Product Component Solutions*. Support for *[SG2] Develop the Design* is provided by the delivery of the Agent Model, Services Model, and Acquaintance Model. Preliminary design is supported through the Services Model where each agent's functionality is documented and in the Acquaintance Model where communication between agents is designed. However, these models fall short of providing support for a detailed design, where more elaborate descriptions including data structures and communication language messages would be defined.

The deliverables out of the Gaia design process provide artifacts for the Technical Data Package described by the subpractice *[SP 2.2-3] Establish a Technical Data Package.* In the Acquaintance Model, a high level description of communication paths is outlined, revealing interface requirements as defined in *[SP 2.3-1] Establish Interface Descriptions.* Gaia provides good analysis coverage of agent plans and actions, but is lacking in the guidance provided for defining inter-agent characteristics (Deloach, Matson, and Li, 2003). No obvious fulfillment of the Product Integration, Verification, or Validation process areas is identified within the Gaia methodology.

RESULTS OF COMPARISON TO CMMI ENGINEERING PROCESS AREAS

The CMMI Engineering category covers activities related to development and maintenance. In this section, each agent-based methodology is mapped to each of the Engineering process areas. For each process area, the specific goals and specific practices supporting each goal have been used as the basis of the comparison.

Comparison to Requirements Management

The practices in the Requirements Management process area describe activities necessary for identifying and controlling changes to requirements. By managing the requirements, the affected plans and data will be kept current. Continual change should be supported, yet should be deliberate and carefully managed (CMMI Product Team, 2002). Table 2 reflects the lack of support for this process area by the methodologies compared.

[SG1] Manage Requirements	MaSE	Gaia
[SP 1.1-1] Obtain understanding of requirements	No	No
[SP 1.2-2] Obtain commitment to requirements	No	No
[SP 1.3-1] Manage changes to requirements	No	No
[SP 1.4-2] Maintain bi-directional traceability	No	No
[SP 1.5-1] Identify inconsistencies between project work and requirements	No	No

Table 2. Requirements Management

Comparison to Requirements Development

The practices in Requirements Development serve to balance the functional requests of stakeholders with what is technically feasible within an agreed upon timeline. The first specific goal of this process area involves working with stakeholders to develop customer requirements (CMMI Product Team, 2002).

In the *[SG2] Develop Product Requirements* specific goal, customer requirements are analyzed along with the operational concept to derive more detailed requirements referred to as the product and product-component requirements. Requirements are allocated to product functions and components that include objects, people, and processes. Additional interfaces may be defined at this state.

During the *[SG3] Analyze and Validate Requirements* specific goal, analysis is conducted to determine the impact that the operational environment will have on the ability to satisfy the stakeholders' needs, expectations, constraints, and interfaces. Customer requirements are also analyzed and validated (CMMI Product Team, 2002).

Analysis revealed no support by either methodology for development of customer requirements. Limited support is provided within MaSE for developing product requirements. Clearly Gaia does not deal with the activities of early requirements engineering (Zambonelli, Jennings, Wooldridge, 2003). As shown in Table 3 and demonstrated through examination, both methodologies fall short in fulfillment of the Requirements Development process area.

[SG1] Develop Customer Requirements	MaSE	Gaia
[SP 1.1-1] Collect stakeholder needs	No	No
[SP 1.1-2] Elicit needs	No	No
[SP 1.2-1] Develop customer requirements	No	No
[SG2] Develop Product Requirements		
[SP 2.1-1] Establish product and component requirements	Yes	No
[SP 2.2-1] Allocate component requirements	Yes	No
[SP 2.3-1] Identify Interface Requirements	Yes	No
[SG3] Analyze and Validate Requirements		
[SP 3.1-1] Establish operational concepts and scenarios	Yes	No
[SP 3.2-1] Establish definition of required functionality	Yes	No
[SP 3.3-1] Analyze Requirements	Yes	No
[SP 3.4-2] Analyze requirements to achieve balance	No	No
[SP 3.5-1] Validate requirements	No	No
[SP 3.5-2] Validate requirements with comprehensive methods	No	No

Table 3. Requirements Development

Comparison to Technical Solution

The Technical Solution process area provides practices to design, develop, and implement technical solutions.
In the *[SG1] Select Product-Component Solutions* specific goal, alternatives are selected based on potential product architecture. Alternative solutions may encompass requirements allocated to different product components.

Designs must provide content for implementation and other phases in the product life cycle including modification, maintenance, and installation. A complete design description should be documented (CMMI Product Team, 2002).

Product components are implemented from the designs established in the previous processes. Implementation includes development and unit testing. Analysis reveals that MaSE has a slightly greater alignment with implementation through its limited tool support. Examination revealed that Gaia does not directly deal with implementation issues (Zambonneli, et al., 2003). Table 4 records that neither of the methodologies examined offer full support for implementation activities.

[SG1] Select Product-Component Solutions	MaSE	Gaia
[SP 1.1-1] Develop alternative solutions and selection criteria	No	No
[SP 1.1-2] Develop detailed alternative solutions and selection criteria	No	No
[SP 1.2-2] Evolve operational concepts and scenarios	Yes	Yes
[SP 1.3-1] Select component solutions	Yes	Yes
[SG2] Develop the Design		
[SP 2.1-1] Design Product or component	Yes	Yes
[SP 2.2-3] Establish technical data package	Yes	Yes
[SP 2.3-1] Establish interface descriptions	Yes	Yes
[SP 2.3-3] Design interfaces using criteria	Yes	Yes
[SP 2.4-3] Evaluate whether product should be developed, purchased, or reused	Yes	No
[SG3] Implement the Product Design		
[SP 3.1-1] Implement Design	No	No
[SP 3.2-1] Develop Support Documentation	No	No

Table 4. Technical Solution

Comparison to Product Integration

Practices in the Product Integration process area enable the development of executable representations of solutions that can be used to demonstrate progress in achieving objectives. Effective management of interface requirements, specifications, and designs help ensure that interfaces are complete and compatible (CMMI Product Team, 2002). The absence of support for Product Integration by MaSE and Gaia is recorded in Table 5.

[SG1] Prepare for Product Integration	MaSE	Gaia
[SP 1.1-1] Determine integration sequence	Yes	No
[SP1.2-2] Establish integration environment	No	No
[SP 1.3-3] Establish integration procedures and criteria	No	No
[SG2] Ensure Interface Compatibility		
[SP 2.1-1] Review interface descriptions for completeness	No	No
[SP 2.2-1] Manage interfaces	No	No
[SG3] Assemble Components and Deliver Product		
[SP 3.1-1] Confirm readiness of components	No	No
[SP 3.2-1] Assemble components	No	No
[SP 3.3-1] Evaluate assembled components	No	No
[SP 3.4-1] Package and Deliver product	No	No

Table 5. Product Integration

Comparison to Verification

Verification practices should be used throughout the software development lifecycle to verify that components are built correctly (CMMI Product Team, 2002). As stated earlier and revealed in Table 6, MaSE provides limited verification guidance with its verification of agent communications. Processes could be formalized using deliverables throughout both of the methodologies as a basis for iterative verification of work products with stakeholders.

[SG1] Prepare for Verification	MaSE	Gaia
[SP 1.1-1] Select work products to be verified	Yes	No
[SP 1.2-2] Establish verification environment	Yes	No
[SP 1.3-3] Establish verification procedures and criteria	Yes	No
[SG2] Perform Peer Reviews		
[SP 2.1-1] Prepare for peer reviews	No	No
[SP 2.2-1] Conduct peer reviews	No	No
[SP 2.3-2] Analyze peer review data	No	No
[SG3] Verify Selected Work Products		
[SP 3.1-1] Perform verification	Yes	No
[SP 3.2-2] Analyze verification results and identify corrective action	Yes	No

Table 6. Verification

Comparison to Validation

Validation practices often involve end users to confirm that results fulfill their intentions when placed in the target environment. Validation of intermediate work products can occur through the life cycle (CMMI Product Team, 2002). With MaSE and Gaia, no direct guidance for validation is presented as shown in Table 7, but the methodologies could be extended

to embrace processes for validation of intermediate work products. The encapsulated design of agent software components is certainly conducive to a modular review in the target environment.

[SG1] Prepare for Validation	**MaSE**	**Gaia**
[SP 1.1-1] Select products to be validated	No	No
[SP 1.2-2] Establish validation environment	No	No
[SP 1.3-3] Establish validation procedures and criteria	No	No
[SG2] Validate Product or Components		
[SP 2.1-1] Perform Validation	No	No
[SP 2.2-1] Analyze Validation results	No	No

Table 7. Validation

CONCLUSION

With its minimal involvement at the requirements development stage, and with the ability to provide more detailed design specifications through agentTool, MaSE offers more alignment with CMMI than Gaia. Both agent-based software methodologies reviewed cover a range of analysis, design, and development activities, fulfilling CMMI practices primarily in the Technical Solution process area. While dissimilar in approach, each methodology examined offers guidance in the design and development of automated solutions to customer requirements. Support for implementation is minimal at best.

Both methodologies could benefit from verification processes which would serve to ensure a more complete and sufficient design. Consistency checking is available with MaSE's agentTool process. With more extensive tool support, the product integration and validate process areas could potentially be addressed. With a primary goal of software engineering being to produce software systems correctly with functionality matching user requirements, verification and validation are recognized as critical to success. The CMMI model places the necessary emphasis in those areas. This analysis has revealed that MaSE and Gaia largely ignore those critical processes.

A summary of the comparison of MaSE and Gaia to the CMMI Engineering process areas is shown in Figure 3. It is clear from these results that fulfillment of CMMI by the agent-based methodologies reviewed has not been fully realized.

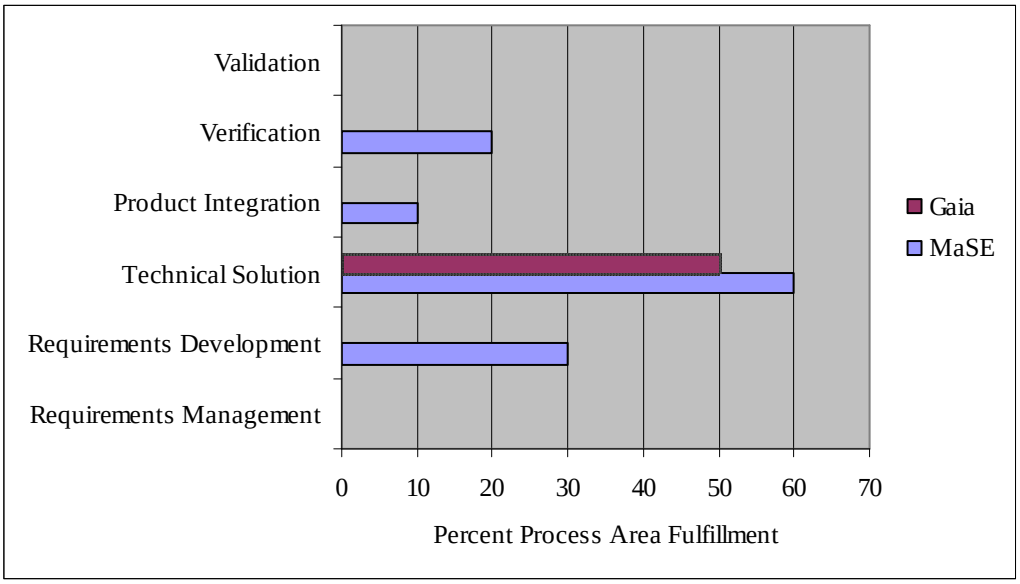

Figure 3. Summary of Methodology to CMMI Comparison

In this paper, only two agent software development methodologies are theoretically compared to the CMMI Engineering process model. Further research is needed to look deeper into the component deliverables within each of the methodologies to identify opportunities for more complete alignment with CMMI. Additionally, conducting a benchmark with projects that are implementing the methodologies under evaluation could provide insight into practices that are essential for efficient agent software development. Those practices could then serve as a basis for comparison. The inclusion of additional agent-based methodologies would likely introduce ideas not represented in the current research.

REFERENCES

1. CMMI Product Team (2002) CMMI[SM] for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing, *Technical Report CMU/SEI-2002-TR-011*.
2. Dam, K. and Winikoff, M. (2003) Comparing Agent-Oriented Methodologies, *Proceedings of the Fifth International Conference Workshop on Agent-Oriented Information Systems*, 3030, 78-93.
3. Dastani, M., Hulstijn, J., Dignum, F., Myer, J. J. (2004) Issues in Multiagent Systems Development, Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), ACM 922-929.
4. DeLoach, S. (2001) Analysis and Design using MaSE and agentTool, *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference*, (MAICS 2001).
5. DeLoach, S., Matson, E., Li Y. (2003) Exploiting Agent Oriented Software Engineering in the Design of a Cooperative Robotics Search and Rescue System, *The International Journal of Pattern Recognition and Artificial Intelligence*, 17 (5).
6. Knublauch, H. (2002) Extreme Programming of Multi-Agent Systems, Proceedings of First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), 1: 704-711.
7. Sudeikat, J., Braubach, L., Pokahr, A., and Lamersdorf, W. (2004) Evaluation of Agent-Oriented Software Methodologies-Examination of the Gap Between Modeling and Platform, *Proceedings from Workshop on Agent-Oriented Software Engineering* (AOSE 2004), 126-141.
8. Wood, M. and DeLoach, S. (2000) An Overview of the Multiagent Systems Engineering Methodology, *Proceedings of the First International Workshop on Agent-Oriented Software Engineering* (AOSE 2000), Lecture Notes in Computer Science, Vol. 1957, 207-222.
9. Wooldridge, M., Jennings, N. and Kinny, D. (2000) The Gaia Methodology for Agent-Oriented Analysis and Design, *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3): 285-312, 2000.
10. Zambonelli, F., Jennings, N. and Wooldridge, M. (2003) Developing multiagent systems: the Gaia Methodology, *ACM Transactions on Software Engineering and Methodology* 12 (3) 317-370.