2005

Comparing Agent Software Development Methodologies Using the Waterfall 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 Waterfall Model

Sarah Bassett Lee
University of Memphis
sblee@memphis.edu

Sajjan Shiva
University of Memphis
sshiva@memphis.edu

ABSTRACT

This paper explores three widely published agent-based software development methodologies, Multiagent Systems Engineering Methodology (MaSE), Prometheus, and Tropos, using the traditional Waterfall model of software engineering as a baseline. Differences between the methodologies are examined and gaps between the agent-based methodologies and the Waterfall approach are identified.

Keywords

Waterfall, Agent Methodology, MaSE, Prometheus, Tropos.

INTRODUCTION

An agent is a computer system that exists in some 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 the flexible, yet complex agent-based software systems. Agent-based development methodologies must be able to support intra-agent design along with inter-agent characteristics. With agent implementations gravitating towards the incorporation of social and cognitive concepts in the design, it is imperative that implementation specifics such as operational environment be considered in the development process (Dastani, Hulstijn, Dignum, and Meyer, 2004).

Methodologies supporting agent-based development have become more popular over recent years. Comparisons of these methodologies have been conducted, but no known research exists that directly compares and contrasts agent-based methodologies using the more traditional Waterfall model as a baseline. This paper will reveal if three widely researched agent-based methodologies are aligned with the Waterfall approach and what, if any, benefits above and beyond are offered.

THE WATERFALL MODEL

The Waterfall model, one of the oldest models in the field of software engineering, is used to manage the software development life cycle, and derives its name from the cascading effect from one phase to another. The model consists of six well-defined phases as shown in Figure 1 (Pressman, 1987). A main idea of the Waterfall Model is that looping back among the phases enables revisions to be incorporated into the lifecycle. When changes are made at any phase, the relevant documentation in other phases should be updated to reflect that change.

In the Systems Engineering phase, the problem is identified, and the goals, objectives, and constraints are specified. In the Analysis phase, the system specification is produced from the detailed definition of the goals, objectives, and constraints that are defined in the previous phase. The specification document that is produced should clearly define the proposed functionality of the system.

In the Design phase, the system specification is translated into system representations such as data structures, algorithms, and system architecture. If the specifications are found to be incomplete, ambiguous, or contradictory, the specification and requirements documents must be revised before proceeding further. The design is translated into software in the Code

phase. Code is written, and testing is performed to make sure that the code satisfies its required specification. If errors are found during testing, then changes and modifications should be done by returning to previous phases.

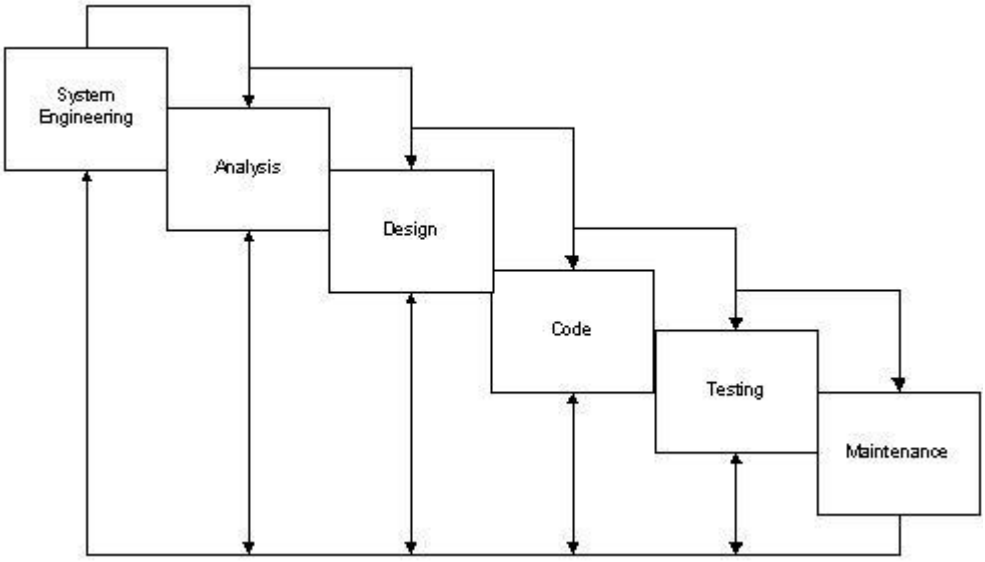

Figure 1. The Waterfall Model

In the phase titled Testing, integration and system testing occurs, where all of the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer for acceptance testing.

The Maintenance phase is usually the longest stage in the software lifecycle. In this phase, software is updated to meet changing customer needs, to accommodate changes in the external environment, to correct errors previously undetected in the testing phases, and to enhance the efficiency of the software.

An advantage of the Waterfall Model is the fact that documentation and quality assurance is present in each phase of the model. A disadvantage is that projects rarely adhere strictly to the complete sequential flow due to conflicts with external time constraints surrounding software development (Pressman, 1987).

ANALYSIS

The creation of inherently complex agent-based software must be supported by adequate processes (Wood and DeLoach, 2000). The Waterfall Model covers a range of activities related to development and maintenance, providing mechanisms for documentation along the way. However, the basic phases of the Waterfall model require extension to address agent-based software development. The Systems Engineering phase is applicable to agent-based software systems development, but must also include consideration of the environment within which the agent will be deployed. The Waterfall design and coding approach must be adjusted to deal with inter-agent activities. To fully support agent development, agent communications must be appropriately designed, coded, and tested for the intended deployment environment.

The primary goal of the MaSE methodology is described as providing a complete lifecycle for design and development of multi-agent systems. Prometheus, another agent-based software engineering methodology that will be compared, is promoted as providing support down to the implementation phase of software development (Sudeikat, Braubach, Pokahr, and Lamersdorf, 2004). A distinguishing characteristic of Tropos is the emphasis placed on early requirements (Bresciani, Perini, Giorgini, Guinchiglia, and Mylopoulos, 2004). In this section, each of the aforementioned agent-based methodologies is examined in terms of the Waterfall phases, uncovering similarities and differences among them.

Examining Multi-Agent Software Engineering

The MaSE methodology was originally developed within a research project at Kansas State University. It has been implemented in multiple projects within academia, including the design of autonomous, heterogeneous search and rescue robots (DeLoach, Matson, and Li, 2003).

MaSE 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 (Wood and DeLoach, 2000). The MaSE approach is an iterative one, with the intention that additional detail is added to documentation across phases in the lifecycle until a complete system design is reached (Cuesta-Morales, Gomez-Rodriguez, and Rodriguez-Martinez, 2004). Practices and deliverables in the MaSE analysis and design phases are depicted in Figure 2.

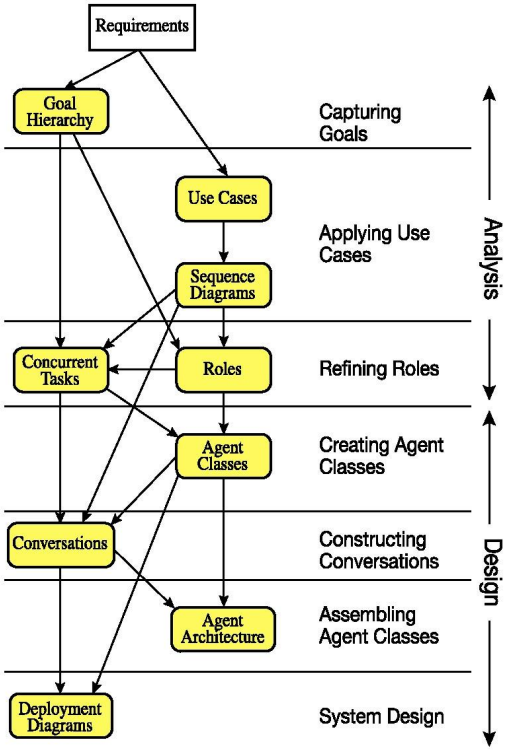

Figure 2. MaSE Phases (Wood and DeLoach, 2000)

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 System Design. The locations of agents within a system are specified in a *Deployment Diagram* (DeLoach, 2001).

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. By evaluating this set of user requirements, the system level goals are defined. The structure of the Goal Hierarchy Diagram enables the analyst to

organize the goals by importance and represent specific goals required for the completion of parent goals (DeLoach, 2001). This aligns with the Systems Engineering phase of the Waterfall model, where the problem is identified and the goals and objectives are specified.

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 (DeLoach, 2001). This scenario building supports the Waterfall Analysis stage, where the system specification is produced.

In the Sequence Diagrams that are produced in the MaSE Analysis stage, the basic communications between agents are defined. In the next step, Refining Roles, each goal is mapped to a role. When roles are fully defined, high-level tasks are created. 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, which supports the Waterfall Analysis objectives.

In the first step of the MaSE Design phase, 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). The conversations and agent architectures that are delivered in these Design stages support the Design step within the Waterfall approach.

In MaSE's System Design step, the overall system architecture is defined, showing the numbers, types, and locations of agents within the system (DeLoach, 2001). This implementation-specific design guidance offered is similar to Waterfall Design specifications.

Strong similarities between MaSE and Waterfall end with the Design step. The agentTool product enables generation of Java code templates, but not implementable code (DeLoach, 2001). The tool does provide some consistency checking capability throughout the phases. It is up to the software designers to author procedures for system, integration, and user acceptance testing of code. Objectives and approach for testing or maintenance are not defined by the MaSE methodology.

MaSE does not support the "construction of plan-based agents that are able to provide a flexible mix of reactive and proactive behavior" (Padgham and Winikoff, 2002). Its treatment of agents seems to be comparable to the consideration of objects, without exploiting the intelligence aspect inherent in agent-based systems.

Examining Prometheus

The Prometheus methodology has been developed in collaboration with Agent Oriented Software, and has been utilized in industry and academia. Prometheus provides guidance from requirements statements through analysis and design in a process of developing models that increase in detail. Prometheus is described as a detailed methodology aimed at non-experts. Its processes are grouped into three main phases as shown in Figure 3: System Specification, Architectural Design, and Detailed Design. Written descriptors are produced at most stages of the methodology, providing design documentation throughout the methodology. Prometheus is designed to be applied iteratively over its three phases (Padgham and Winikoff, 2002).

In the System Specification phase, the basic functionalities of the system are identified along with inputs and outputs. One of the main activities involves determining the proposed system's environment. Percepts, incoming information from the environment, and actions, means by which an agent affects its environment, are identified. Use Case Scenarios are used to describe goals and how they will be achieved.

The Architectural Design phase identifies the agents that will be part of the system and how those agents will interact. Agent types are derived by grouping functionalities, considering coupling and cohesion. For each agent type, a lifecycle is defined which includes how and when the agent will be initialized and destroyed and what its functionality will be. Data used and produced by the agent type along with events that the agent should respond to are identified.

One of the most important deliverables of Prometheus is produced in the Architectural Design phase. The *System Overview Diagram* provides a general picture of how the system as a whole will function, including agent types, communication links, and data. Also during this phase, interaction protocols are developed.

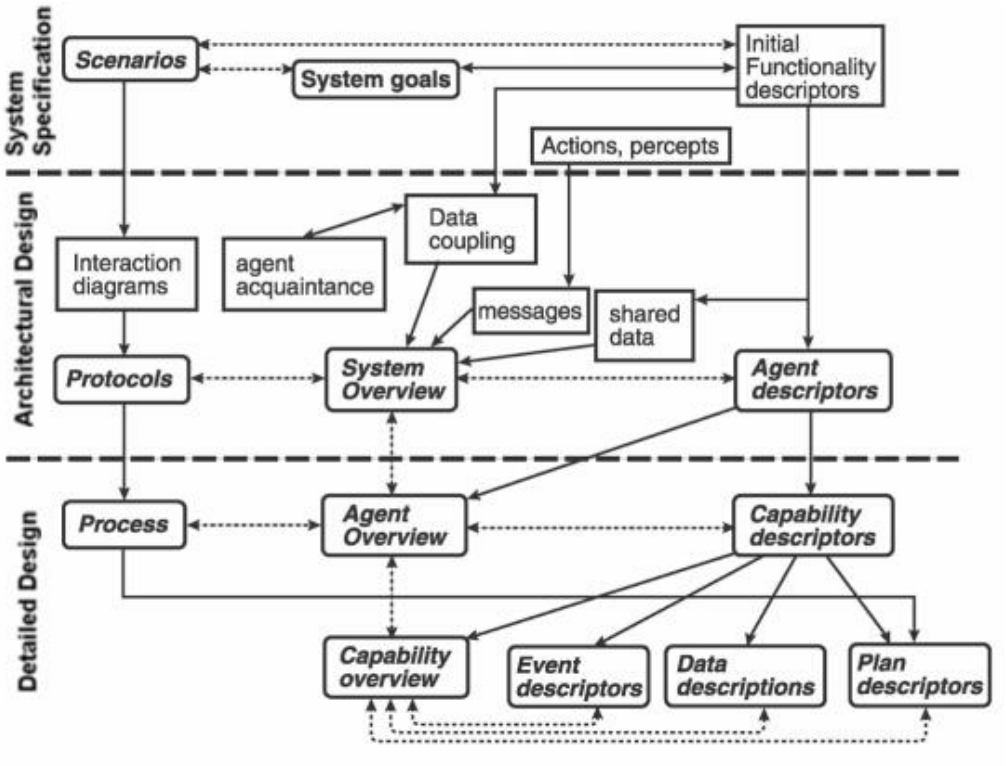

Figure 3. Prometheus Phases (Padgham and Winikoff, 2002)

In the Detailed Design phase, the details of how the agent will accomplish its tasks are defined. This phase provides a high-level view of the components within the architecture along with interactions between them. Detailed data structures are defined and a top level view of each agent's internals is depicted. The results of this phase includes a collection of event-triggered plans, which can be mapped directly to implementation constructs provided by platforms such as JACK™ Intelligent Agents, an agent-oriented development environment that is fully integrated with Java.

The System Engineering and Analysis phases in the Waterfall model are represented in the System Specification phase of Prometheus, where goals are determined and specified. Identifying information coming into the environment and effects going out to the environment from the agent are an integral part of this phase.

In the Architectural design phase, agents are grouped depending on their functionality. The System Overview Diagram provides a general picture of how the integrated system will function, tying together agents, events, and shared data objects (Padgham and Winikoff, 2002). The final phase in Prometheus is the Detailed Design phase where the internal design of each agent is determined. Architectural and System Design within Prometheus support the objective of producing an implementation-specific design presented by the Waterfall approach.

While different from MaSE in its approach to developing agent software, the activities within Prometheus are similar to phases of the Waterfall model. Comparison of Prometheus to the Waterfall approach reveals that it is an expansion of the traditional model, yet extends the scope to include specialization for developing agent software. Prometheus supports the belief, desire, and intention (BDI) agent architectures in its detailed design. A BDI architecture considers agents to have beliefs about itself and its environment, desires about future states, and intentions or plans about its own future actions. The Prometheus design produces a plan-based system where agents react to events, based on their beliefs about the situation (Padgham and Winikoff, 2002).

The Prometheus Design Tool (PDT) allows for entry and editing of a design, and provides cross checking to help in maintaining consistency of the system design. The JACK development environment (JDE) provides a drag-and-drop graphical user interface for building the agent system, and supports the artifacts from Prometheus's Detailed Design phase. Agents defined within the JDE generate JACK code that can be compiled and run (Padgham and Winikoff, 2002).

Procedures for unit and integration testing of that code are left to the designers to define. There are no explicit guidelines provided by the Prometheus methodology for testing or maintenance.

Examining Tropos

The Tropos methodology was initially developed at the University of Toronto, and has been implemented in several projects within academia (Sudeikat, et al., 2004). Tropos focuses on early requirements analysis and is a goal-driven approach to software development. There are five main phases within Tropos, depicted in Figure 4: Early Requirements, Late Requirements, Architectural Design, Detailed Design, and Implementation. Tropos is based on the BDI agent architecture, and the notions of belief, desire, and intention are formally specified in the modeling activities (Bresciani, et al., 2004).

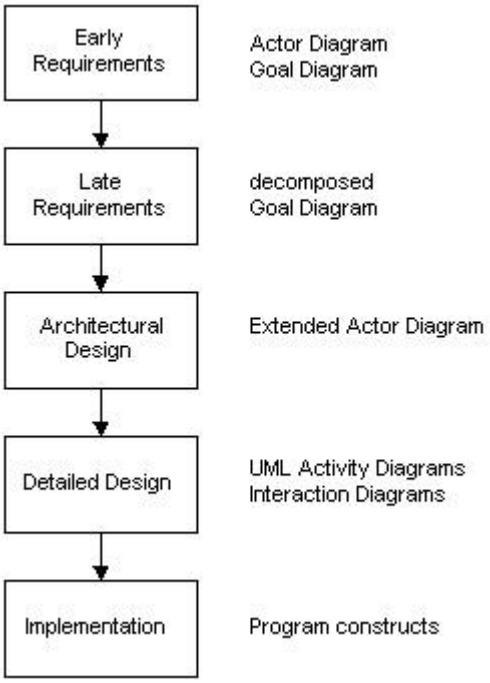

Figure 4. Tropos Phases

In its Early Requirements stage, Tropos uses the concept of actors and roles to model stakeholder intentions to build an understanding of the problem to be solved. Two types of goals are identified at this stage. *Hardgoals* lead to functional requirements, and *softgoals* are related to non-functional requirements. Deliverables of this first phase include an *Actor Diagram* and a *Goal Diagram*. The Actor Diagram depicts how actors depend on one another for goals to be accomplished, plans to be executed, and resources to be supplied. In the Goal Diagram, the analysis of goals and plans for responsible actors are depicted.

The Late Requirements stage extends the models created in the previous phase and models the target system within its environment. The proposed system is represented as an actor that has dependencies with other actors within the organization. These dependencies represent the functional and non-functional requirements of the system. Analysis in these first two stages is intended to provide the context within which the proposed system will be designed.

The Architectural Design phase describes the system capabilities, and these capabilities are grouped to form agent types. These grouping are outlined in an *Extended Actor Diagram*. In Detailed Design, each architectural component is defined in more detail. The detailed agent specification is delivered through *UML Activity Diagrams*, which represent capabilities and plans, and *Interaction Diagrams* showing the interactions between each agent.

The goal of the Implementation phase is to generate program constructs from the detailed design using an agent-oriented programming platform (Dam and Winikoff, 2003). The platform specified for implementation with Tropos is JACK™

Intelligent Agents. There is a direct correspondence between the detail design specification and the constructs provided by JACK (Bresciani, et al., 2004).

Tropos divides the Requirements phase into early and late requirements, where goals are first analyzed and then divided into sub-goals. The Early Requirements Phase of Tropos is concerned with understanding the problem through studying the goals and relevant actors of the proposed system. In Late Requirements, the proposed system is described in terms of its operational environment (Mylopoulos, Castro, and Kolp, 2000). Both requirements phases map closely to the System Engineering and Analysis stages of the Waterfall approach.

Similarly, the Architectural Design phase of Tropos, where the global architecture is defined in terms of capability groupings that are interconnected through data and control flows, is comparable to the Design step of Waterfall. The Detailed Design phase further exploits the Waterfall Design stage in that implementation details are defined at the component level. The concepts in Detailed Design are oriented towards the JACK implementation platform (Bresciani, et al., 2004).

Tropos is clearly a methodology with phases comparable to the Waterfall Model. It is distinguishable from MaSE and Prometheus in that it puts more emphasis on the requirements phase than any other phase in the life cycle. With Tropos, a closed multi-agent system is assumed (Dastani, et. al, 2004). For software systems where agents will have advanced reasoning for plans, goals, and negotiations, Tropos falls short of providing guidance for detailed design (Bresciani, et al., 2004).

Limited consistency checking is offered by JDE to support propagation of changes across phases. Like MaSE and Prometheus, Tropos does not provide explicit guidance for testing and maintenance of generated code

CONCLUSION

None of the three agent methodologies provides extensive support for the entire life-cycle of software system development as depicted by the Waterfall Model. Each of the methodologies covered the Waterfall Analysis and Design objectives. The coding phase is supported on some level by each of the methodologies; however testing and maintenance are not.

Certainly the iterative approach of the Waterfall model could be applied to each of the agent-based methodologies. The component-based nature of multi-agent systems lends itself to iterative, modular design and development. Graphically based modeling tools supporting the agent-based methodologies enable phases to be visited iteratively, with changes to documentation in any phase contributing to a complete and consistent design.

MaSE and Prometheus have more extensive tool support and consistency checking. This level of tool support is currently lacking in Tropos. The agentTool product that supports the MaSE methodology enables the generation of Java code templates, but not implementable code.

Both Prometheus and Tropos support BDI agent architecture, extending their specialization beyond traditional software engineering. Prometheus provides a more detailed process for detailed design of intelligent agents, and is particularly suited for BDI type systems (Padgham and Winikoff, 2002). The modeling notation within Tropos is not sufficient for representing reactive tasks or inter-agent communications protocols (DeLoach, et al., 2003). MaSE is perceived to be a more general approach to software engineering in that it does not exploit agent characteristics.

Tropos provides guidance for the early phases of software systems development through detailed design. Tropos is significantly different from MaSE and Prometheus in its support for early requirements. Prometheus provides more detailed guidance in its architectural design phase than Tropos and MaSE with its production of the System Overview Diagram. Extension of Prometheus to include more analysis of requirements would be a logical evolution of the methodology and would extend its usefulness and completeness in guiding the entire lifecycle of software development.

In this paper, three agent software development methodologies are compared using the Waterfall model as a baseline. Activities within each methodology are grouped in Table 1 according to the Waterfall phases. The analysis has revealed strengths and weaknesses within each of the methodologies, and could support the selection process for choosing an appropriate methodology for agent-based systems development. While Prometheus is revealed to be the most mature of the three in its specialization for agent-based software development, Tropos is the only one offering guidance in requirements analysis. For a project requiring extensive requirements engineering, Prometheus and MaSE would be less than ideal.

Including additional agent-based methodologies in the comparisons and including popular object-oriented approaches as baselines for comparison would likely introduce findings not represented in the current research. A controlled implementation of the methodologies under comparison would provide additional factual insight into the strengths and weaknesses of each.

Waterfall	MaSE	Prometheus	Tropos
Systems Engineering Identify goals, constraints, objectives	- Capture goals - Apply Use Cases to extract main scenarios	Determine environment	Goal analysis
Analysis Detailed system specification derived to clearly define proposed functionality	Refine roles, creating roles responsible for goals	Determine goals and how to achieve them	- Actor plans and interactions - Decompose goals into subgoals
Design Translate system specification into data structures, algorithms, and system architecture	- Create agent classes, mapping roles - Construct conversations - Assemble agent classes, identifying internal functionality - Specify location of agents within the system	- Define agent types - Design system structure - Define agent interaction protocols - Define capabilities, internal events and plans - Define detailed data structure	- Identify which events are generated and received - Form agent types from capability groupings - Define capabilities and plans of each agent - Describe interaction of each agent - Map concepts to language constructs
Code Translate design into software	Java code templates produced	JACK implementation platform	JACK implementation platform
Testing Perform integration and system testing	No explicit guidance provided	No explicit guidance provided	No explicit guidance provided
Maintenance Update software	No explicit guidance provided	No explicit guidance provided	No explicit guidance provided

Table 1. Methodology Comparisons

ACKNOWLEDGEMENTS

The authors acknowledge the contributions of Alaa M. Alayssh and Chandler Ford in preparation of this paper.

REFERENCES

1. Bresciani, P., Perini, A., Giorgini, P., Guinchiglia, F., and Mylopoulos, J. (2004) TROPOS: An Agent-Oriented Software Development Methodology, *In Journal of Autonomous Agents and Multi-Agent Systems 8(3): 203-236.*
2. Cuesta-Morales, P., Gomez-Rodriguez, A., Rodriguez-Martinez, F. (2004) Developing a Multi-Agent System using MaSE and JADE, *UPGRADE* (August 2004), Vol. 5 No. 4: 27-32.
3. 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.
4. Dastani, M. and Hulstijn, J. and Dignum, F. and Meyer, 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.
5. DeLoach, S. (2001) Analysis and Design using MaSE and agentTool, Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference, (MAICS 2001).
6. 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).
7. 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.
8. Mylopoulos, J., Castro, J., and Kolp, M. (2000) Tropos: Towards Agent-Oriented Information Systems Engineering, *Position Paper at the Second International Bi-Conference Workshop on Agent_Oriented Systems,* (AOIS2000).

9. Padgham, L. and Winikoff, M. (2002) Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents, *In proceedings of the workshop on Agent-oriented methodologies at OOPSLA 2002*.
10. Pressman, R. (1987) *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Inc., New York.
11. 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.
12. 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.