User and System Stories: An Agile Approach for Managing Requirements in AOSE

Sebastian Rodriguez
RMIT University
Melbourne, Australia
sebastian.rodriguez@rmit.edu.au

John Thangarajah
RMIT University
Melbourne, Australia
john.thangarajah@rmit.edu.au

Michael Winikoff
Victoria University of Wellington
Wellington, New Zealand
michael.winikoff@vuw.ac.nz

ABSTRACT

The agile software development life cycle is widely used in industry today due to its highly flexible and iterative processes that facilitate rapid prototyping. There has been recent work in bringing concepts and processes from agile methodologies to agent-oriented software engineering (AOSE). We contribute to this effort by presenting in this paper a novel approach to capturing requirements of agent systems in AOSE using and extending agile concepts. In this paper, we propose to adopt and extend the well-known concept of *User Stories* to facilitate the development of agent systems. We introduce a novel concept, *System Story*, that defines requirements from the perspective of the system. These System Stories are refinements of User Stories and provide more intuitive mappings to agent concepts in the design and implementation. We show how our approach allows better traceability of requirements between stories and the different software development artifacts. We validate our proposal with a feature-based comparison to recent related work, and a preliminary user evaluation based on a drone simulation of a simple search and rescue case study.

KEYWORDS

AOSE; Engineering MAS; Agile methodologies

ACM Reference Format:
Sebastian Rodriguez, John Thangarajah, and Michael Winikoff. 2021. User and System Stories: An Agile Approach for Managing Requirements in AOSE. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

The agile software development life cycle (SDLC) is widely used in industry today. In 2017, 71% of the organizations consulted in [27] responded that they use agile approaches in at least some of their projects. The agile approach encourages a flexible and iterative process that generates rapid prototypes that are continuously refined. This involves much more interaction and collaboration with the users and developers of the software system. The core values expressed in the agile manifesto [2] are at the center of this success.

There has been recent work in bringing concepts and processes from agile SDLC to agent-oriented software engineering (AOSE) [5, 21, 22, 31, 33]. We contribute to this effort in this paper - we present a novel approach to capturing requirements of agent systems in AOSE using and extending agile concepts.

A User Story is a brief informal description of a feature described from the perspective of a user of the system. For example, for a bookstore system: *As* a customer, *I want to* search for books by keywords, *so that* I can find the book I want.

Some of the key benefits of using User Stories are that they [8]: are comprehensible by both the users and the developers; enables better scoping of requirements; encourage deferring details until a better understanding of the needs are established; and promote a greater understanding of the domain specific concepts and logic by the system developers.

In current agent development methodologies such as TDF [16], Prometheus [25], Tropos [4], O-MaSE [11], INGENIAS [26], and ASPECS [10] the process of managing requirements from elicitation to implementation follows the more traditional way of developing software, rather than the more modern agile approaches.

Most of the agent methodologies use concepts and techniques that are *foreign* to the domain experts and end users. The initial stages of gathering the requirements of the system are often cumbersome and require extensive modelling to map the requirements to suitable artifacts of the respective methodologies.

In this article we present an approach that not only adopts the concept of User Stories, but also extends the concept to *System Stories*, which capture features from the system's perspective, considering the system as a first class citizen. In our approach, we refine User Stories to System Stories. For example, w.r.t to the User Story above, a System Story might be: *As* the System, *I want to* index all the books by keywords, *so that* I can retrieve them by keyword. These System and User Stories are mapped to AOSE design artifacts and corresponding constructs in an agent programming language. Although there has been recent work on agile AOSE methodologies (e.g. [5, 33]), they do not include System Stories or acceptance criteria, the conditions under which the system behaviour can be accepted by the stakeholders. Our work, by including both these aspects, provides a richer approach with better traceability.

We hypothesize that the agile approach we propose will provide a more natural and flexible way to managing requirements that are: (i) easy to understand; (ii) mapped to the agent design concepts that realise them; (iii) easier to maintain; and (iv) allow traceability from execution to requirements.

Our approach is general and complementary to existing AOSE methodologies. For the purpose of illustrating our approach we use in this paper the TDF (Tactics Development Framework) agent design methodology [16] and the SARL agent implementation platform [29]. The choice of these tools are due to the fact that they are used for an industry project that necessitates this work.

In this paper we: (i) present an agile approach to capturing the requirements via User and System Stories (USS); (ii) show how USS

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

map to agent concepts in AOSE; (iii) describe a traceability framework with supporting tool; (iv) present a feature-based comparison of USS to other approaches; and (v) provide a proof-of-concept evaluation of the above mentioned hypotheses with users from our current industry collaborations.

2 BACKGROUND

In this section we introduce the preliminaries in terms of User Stories and acceptance criteria as used in agile approaches, and a brief overview of TDF and SARL.

2.1 Agile & User Stories

Agile approaches finds their roots in the movement started by the Agile Manifesto [2] that defines its core values and principles. A number of software development life cycles (SDLC) have embraced the fundamental principles, e.g. Scrum [30] and Extreme Programming [1]. A key value of agile approaches is the close collaboration of the development team with stakeholders. Agile methods always favor and encourage *conversations* to develop a shared understanding over rigid requirement specification documents. A number of techniques have been proposed to understand the objectives and expectations of the stakeholders (e.g User Interviews, Questionnaires, Story-Writing workshops) [8]. In this context, the most common technique to gather and document requirements in agile frameworks are *User Stories* [1]: a brief informal description of a feature described from the perspective of a user of the system.

Writing good User Stories and capturing sufficient detail is still a matter of debate and often requires specialized training. However, there is a general agreement that stories should remain high level in the early stages [8, 9]. They are then refined and detailed as they get a higher priority in the project development.

While there is no single format of User Stories, the most widely used template is: *As (role), I want to (do something), so that (reason).* For example, for a bookstore system: *As* a customer, *I want to* search for books by keywords, *so that* I can find the book I want.

The User Stories contain as well *Acceptance Criteria* (AC), which are a set of statements that identify the conditions under which a system behaviour can be accepted by the user or stakeholders. A common format for AC is the scenario-oriented *Given/When/Then*, derived from Behavior Driven Development (BDD) [24]. The essential idea is to break down the scenario into three main sections: (a) *Given* - describes the state of the world before the system executes the behavior being described; (b) *When* - identifies the triggers of the behavior; and (c) *Then* - details the expected outcomes of that behavior. For example, *Given* a set of keywords, *when* I search for a book, *then* books matching those keywords are displayed.

2.2 TDF & SARL

There are many agent-oriented software engineering (AOSE) methodologies to help software engineers to conceptualise, design, and implement agent-based systems. MaSE [11], Tropos [4], INGENIAS [26], Gaia [35], PASSI [6], Prometheus [25], and TDF [16], are some of the most commonly used AOSE methodologies for developing agent-based systems. Although these methodologies differ in many ways, they follow the common software engineering core activities of: specification, design, implementation, and testing. The requirements specification approach described in this paper via User and System Stories is intended to complement the existing AOSE methodologies rather than replace any of them. For the purpose of grounding our work we chose the TDF methodology as it is used in the industry project that motivates this work.

TDF [15, 16] is the successor to Prometheus [25], a mature and popular AOSE methodology. A pilot study has shown that TDF significantly improves comprehension of behaviour models, compared to UML [18]. The methodology follows 3 phases in an iterative manner: *System specification* where the system-level artifacts are identified, namely goals, scenarios, percepts, actions, data, actors and roles; *System architecture* where the internals of the system are specified, namely the agents that enact the different roles and the interactions between them; and *Detailed design* that defines the internals of the agents, namely plan diagrams, tactics and internal messages/sub-goals.

There are several agent implementation frameworks that support the implementation of intelligent agent systems, e.g. JACK [34], SARL [29], Jadex [28], and Jason [3]. We use SARL in our work, since it is what is currently used in our industry project and the TDF design artifacts are readily implemented in SARL (recent work [18] also used this combination).

SARL is a general-purpose agent-oriented system implementation framework that provides the fundamental abstractions to develop and deploy distributed complex systems. Due to its generic and highly extensible architecture, SARL is able to integrate new concepts and features quickly and has been adopted by a number of academic and industrial institutions to develop a wide range of applications. Space preclude a more detailed introduction to SARL, which is not required for this paper. For further details see [29].

3 APPROACH

In this section we describe our approach to specifying the requirements of the system via User and System Stories and how they translate to agent concepts, resulting in all of the benefits described in the Introduction of this paper. Our approach is general and does not necessitate any particular development process *per se.* (e.g. Scrum, Kanban, etc.).

Given a high level specification of the system in terms of *objectives*, our approach comprises the following steps:

(1) for each objective, identify User Stories using classical techniques (see [8]);
(2) refine each User Story into system level requirements in the form of System Stories and their acceptance criteria; and
(3) during the development process (defined by the selected SDLC and generally iterative):
    - map the System Stories, including the associated acceptance criteria, to the relevant agent concepts.
    - maintain a process ledger for the purpose of traceability.

We will describe each of the above steps in detail in the following subsections. In the remainder of this paper, we will use the term ‘story’ when the discussion applies to both User and System Stories.

Search and rescue case study and testbed: In order to illustrate our approach we will use an example case study: a search and rescue domain where autonomous drones assist humans to locate and assist disaster victims. The drones assist in locating and identifying

Objectives	**User Story**		**System Story**		***As***	***I want to***	***So that***	**Technology Opportunities**	**Ethical Consideration**
Explore	1	Explore Area			Drone Operator	assign to drones areas to explore	They find victims and notify me.		
Explore			1.1	Find Victims	Drone	explore an area assigned to me.	I can find victims.	Path Planning Exploration	
Explore			1.2	Detect Victims	Drone	detect victims	I can locate their position	Computer Vision; Infrared Sensors	Potential Bias, Accuracy
Explore			1.3	Locate Victims	Drone	locate victims	I can inform operator	GPS; Galileo; Indoor Positioning	
Intervene	2	Assist Victim			First Responder	reach victims	I can render appropriate aid		
Intervene			2.1	Inform First Responder	System	use Drone images to identify victim	I can provide First Responder with personal medical data	Computer Vision	Data Privacy
Intervene			2.2	Guide First Responder	System	use Drone mapping to guide First Responder to victim	First Responder can reach victim safely and quickly	Path Planning	
Intervene			2.3	Signal First Responder	Drone	visually signal to First Responders a victim's location	First Responder can find the victim faster		

Table 1: Sample User and System Stories from Drone Case Study

victims, via tasks assigned to them by the *human drone operator*, which they carry out autonomously. When victims are located and identified, human *first-responders* assist the victims. Therefore, there are interactions between the drones and the humans. There are two primary objectives: *exploration* in order to search for victims; and *intervention*, providing assistance to any victims found.

3.1 Identifying User Stories

Consider the objective of exploring to identify victims in our case study. A User Story for the drone operator might be: *As* drone operator, *I want to* to be able to assign to the drones areas to explore *so that* they can locate victims and notify me. This follows the *Who/What/Why* format as described in §2.1. We can represent these User Stories as shown in Table 1.

There are well-developed existing techniques for gathering, prioritizing, and scoping User Stories [8], and we propose to simply use them for this step in our approach.

When identifying User Stories, it is also important to identify the acceptance criteria for that User Story, which identifies the expected behaviour of the system in different scenarios. As mentioned in §2.1 these are typically in the form *Given/When/Then*. Since there are already well-developed and accepted techniques for identifying and deriving acceptance criteria [8], we simply propose to use these. Table 2 shows some example acceptance criteria for the User Story 1 that shows the expected behaviour of the drones in situations where the drone is idle and busy, respectively: if the drone is idle, then when receiving a request to explore an area, it commences exploring the area immediately. However, if the drone is already exploring, then instead it adds the new area to its exploration queue.

3.2 Extracting System Stories

Whilst the standard agile approaches would take the User Stories and map them to design and implementation of the system in an iterative fashion, we introduce the novel concept of System Stories which further refine the requirements enabling them to be more easily mapped into agent concepts.

System Stories. The basic intuition is that each User Story is refined into one or more System Stories. The System Stories associated with a given User Story consider the requirement from the system's perspective, and expand on the functionality that the system needs to have, in order to be able to meet the user's needs.

We represent System Stories using the same Who/What/Why format as User Stories. For example, given User Story 1 (see Table 1), we could firstly define a System Story (story 1.1) along the lines of *As* Drone, *I want to* explore an area assigned to me, *so that* I can identify victims. Note that here we changed the perspective from the user to system.

The process of developing System Stories involves two techniques. Firstly, as illustrated above, we take a User Story and convert it into a System Story by changing the role, and shifting responsibility for the requirement to the system (or part of it). Secondly, we refine System Stories by breaking them down, asking "how might this be done?". For example, finding victims may require detecting humans, and then confirming their location.

Table 1 shows example refinements for our scenario. The *so that* column captures the rationale (or, alternatively, explanation) for each system goal. In some cases the rationale for a given System Story relates to its associated User Story. For example, the drone identifies humans so that it can inform the operator. In other cases, it may relate to a subsequent System Story. For example, the drone explores an area (System Story 1.1) so that it is then able to locate humans (System Story 1.3). Essentially, new System Stories can be derived from other User Stories or System Stories, which creates a hierarchy of requirements.

It is also possible to describe System Stories at the early stages of the SDLC without identifying the specific roles (or agents) within the system by using the generic term 'System' as in Story ID 2.1 in Table 1. As the system is refined further the generic term can be replaced with a specific role (or agent). In the process of defining both User and System Stories, it is important to recall that, as discussed in [8], stories need to remain negotiable and we should resist the temptation of including too much detail upfront, in the details of the User Story or derived Systems Stories. Stories should remain as reminders to have a conversation.

Acceptance Criteria. As with a User Story, when defining a System Story we identify a set of acceptance criteria, which will be

Story ID	Scenario	Actor	*Given*	*When*	*Then*
1	Drone Idle	Drone Operator	the drone is at base	it is assigned an area to explore	it starts exploring autonomously
	Drone Busy	Drone Operator	the drone is exploring	it is assigned an area to explore	it adds the area to its exploration queue
1.1	High Priority	Drone	A high priority area X	the area X is assigned to me	I employ a Plow Sweep
	Low Priority	Drone	A low priority area Y	the area Y is assigned to me	I employ a Random Walk

Table 2: Sample Acceptance Criteria for Stories defined in Table 1.

used for testing and validation purposes. As in the previous section, we propose to use existing techniques [8]. Table 2 shows some sample acceptance criteria for System Story 1.1 from Table 1. For example, in the situation where a high priority area is assigned to a drone, it should employ a Plow Sweep, as opposed to a Random Walk (these are different exploration strategies). Note that the expected behaviours of the Plow Sweep and Random Walk can also be defined as acceptance criteria at later stages of the SDLC prior to the implementation of these techniques.

Technology opportunities and ethical considerations. As System Stories are designed, various specialised technological solutions may need to be considered in the system development stage to realise these requirements. For example, story 1.1 may require strategies for exploration and path planning techniques; and story 1.2 may require computer vision techniques to identify humans. The other critical aspect in designing any intelligent system that incorporates AI technology, is the ethical considerations [13, 32]. Considering the ethical issues at the requirements stage allows a more systematic and thorough manner of capturing them which also affords traceability at later stages. In our case study, story 1.2 should consider the potential bias in the computer vision techniques in identifying victims, and story 2.1 needs to consider data privacy issues. We document both the technology opportunities and ethical considerations in our template as shown in Table 1.

3.3 System Story Mappings and Traceability

The User and System Stories described above are typically defined during the requirements gathering meetings[1] where the stakeholders and the system development team discuss and develop a shared understanding of the system's features. During the system development phase, the development team will have to translate those stories into appropriate design artifacts and executable code making sure that all expected behaviors are met. In this section, we provide some guidelines on how this translation could be carried out. Additionally, we show how to maintain records of these mappings for traceability purposes. As mentioned, we ground our approach in the TDF design methodology and SARL agent implementation language, though the general concepts (e.g. goals, agents, plans, actions, etc.) are adaptable to other approaches[2].

Translating Stories. The system design and architecture is a critical part of the SDLC for the long term maintainability of the system and for the productivity of the development team [19, 20]. The design is then translated into implementation artifacts.

In our approach we map stories to both agent design and implementation artifacts. We can intuitively relate the Who/What/Why

[1]The events will take different names according to the methodologies (e.g. Sprint Planning, Backlog Grooming, etc.)

[2]See [12] for similarities between some AOSE methodologies.

Story Element	Story type	
	User	*System*
Who (As)	<agent/role/system module>	<agent/role/system module>
What (I want to)	<user input / percept>	<achieve or maintain goal> <do action>
Why (so that)	<system output / action>	<achieve or maintain goal> <handle perception>

Table 3: Story mapping templates

Story ID	Story Element	Story Line	$\Rightarrow$	Agent Concept	Instance
1	Who (As)	Drone Operator		User	Operator UI
	What (I want to)	assign to drones areas to explore		Percept	Area Assignment
	Why (So that)	They find victims and notify me		Action	Inform Operator
1.1	Who (As)	Drone		Agent	Drone
	What (I want to)	explore an area assigned to me		Goal	Explore Area
	Why (So that)	I can find victims		Goal	Find Victim
2.1	Who (As)	System		Agent	C2 System
	What (I want to)	use Drone images to identify victim		Goal	Identify Victim
	Why (So that)	I can provide F.R. with medical data		Goal	Obtain Medical Data

Table 4: Stories to agent concepts mapping.

format of a story to agent concepts using the generic template shown in Table 3. Using the template, we can then create the mappings depicted in Table 4. Note that we identify a particular *instance* for each agent concept in the mapping process. Note that the "what" part of a system story can, in the case where the system story involves performing a single action, be mapped to an action, rather than a goal. Similarly, the system story's "why" can sometimes be to respond to a percept.

The User Stories in general help identify the inputs (percepts) and outputs (actions) of the system. For instance, from User Story 1 we can know that the *Operator UI module*[3] will trigger an input (percept) *Area Assignment* that will generate the intention (goal) *Find Victim* and produce the output (action) *Inform Operator.* Figure 1(a) shows the TDF architecture overview diagram illustrating a sample of the concepts associated with the the User Stories.

Using the System Stories related to the User Stories, we can further decompose the high level goals into subgoals. For instance, we can deduce that in order to *Find Victim* (from User Story 1) the agent is required to perform other goals such as *Explore Area* and *Locate Victim* (from System Story 1.1). Following this approach

[3]An alternative design could represent the Operator as an agent that represents the human user in the agent system

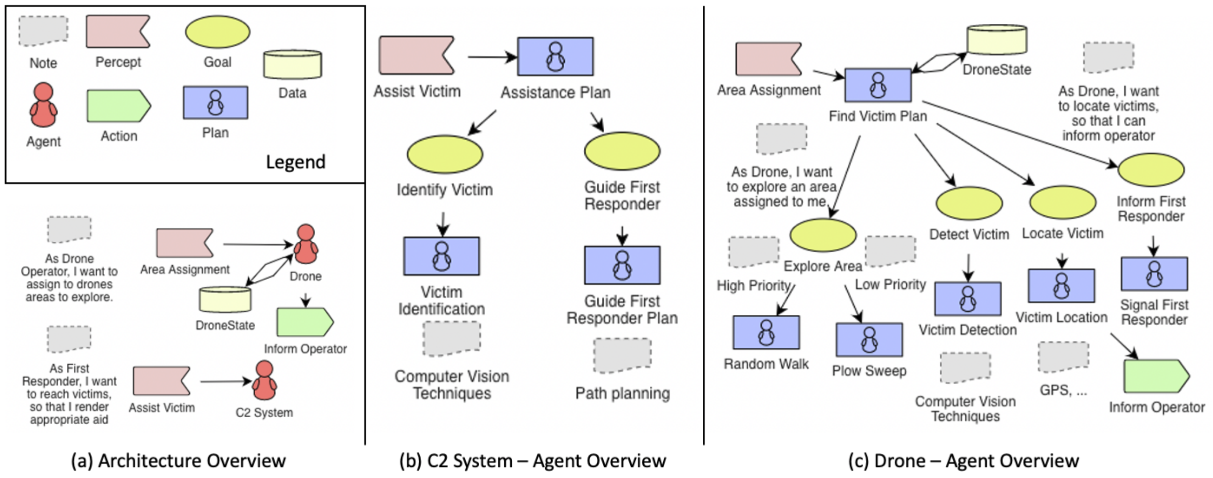


Figure 1: TDF System Design Extract

we start to identify dependencies and hierarchies of goals and plans. Note that these initial goals can be later refined into more convenient ones when needed.

The System Stories therefore enable the development of the agent whose perspective the story captures. Figures 1(b) and 1(c) shows a sample of the agent overview diagrams of the *C2 System* ("command and control system") and *Drone* agents, respectively. Note that for the *Drone* agent, the goals *Detect Victim, Locate Victim* (from System Story 1.2 and 1.3) and *Inform First Responder* (from System Story 2.1) are merged as steps of the *Find Victim* plan.

System stories are related (and typically mapped to) goals, but differ in that they can also be mapped to an action and/or a percept (see Table 3), and that a system story identifies both the "what" and the "why".

Most agent design methodologies support the translation from design to implementation. The detailed agent diagrams in TDF are readily translated to implementation constructs in agent implementation languages such as JACK [17] and SARL [15].

Translating Acceptance Criteria. As mentioned, User and System Stories provide the initial mappings to define the architecture of the agents, which may be incomplete. For instance, System Story 1.1 identifies that Drones should be able to *Explore Area*, but nothing is detailed on how to handle that goal in different scenarios. We use *Acceptance Criteria* for the purpose of refining the details.

When examining the acceptance criteria in Table 2 for Story ID 1.1, we notice that the Drone has two options to explore: (a) Random Walk or (b) Plow Sweep. Further, we can define the conditions under which each option is chosen based on the priority of the area to explore: low or high. This is translated in the design as two different plans to achieve the *Explore Area* goal as shown in figure 1(c).

Thus, acceptance criteria enables the identification of plans for the associated System Story, as a first iteration in the design process. More detailed acceptance criteria will further detail the plans for the different goals. For instance, to answer questions such as: what is the separation between lines in a plow sweep? how long should the drone do random walk of the assigned area? A template to map acceptance criteria to appropriate agent concepts is then:

Scenario	⇒ a plan that handles a goal related to the associated story.
Given	⇒ the context that describes the applicability of the plan.
When	⇒ the trigger of the plan (goals, messages and percepts).
Then	⇒ the expected outcomes of the plan.

The above clear mapping is one of the reasons for choosing the Given/When/Then format for representing acceptance criteria. Although the primary mapping of acceptance criteria is to establish plans to achieve goals, they can also provide other information such as insights into the **belief sets** the agent will need access to. Expanded versions of the System Stories for our case study will include other conditions like battery level, time of exploration, etc. that gives rise to the need for a *DroneState* belief set to store this information. Additionally, acceptance criteria should be used as a basis for the generation of tests, for example see [5] for a behaviour driven development (BDD) approach.

Traceability. The process of translating the User and System Stories into the agent design artifacts as described above results in a single story mapped to different parts of the design and the designers may further refine them in the design and indeed in the implementation. Further, it is also common practice in any SDLC, for different development teams to work on the different phases according to the project needs, their experience, preferences, etc. This results in the distribution of the rationale of design and implementation choices. This distribution, whilst necessary, makes it difficult to maintain the evolution of the requirements over time and to trace the behaviors of the system back to the stories. For these reasons, we propose a requirements tracing framework to keep track of the changes and concepts derived as described below.

The main concepts of the framework (inspired by [23]) are presented in the metamodel shown in Figure 2. At its core, is a *Process Ledger* of traces, where each *Trace* is a mapping between the concepts of the different artifacts used during the SDLC. A *SDLCArtifact* represents any conceptual artifact used in a particular phase of the SDLC. For example, *Story* is a SDLCArtifact of the requirements

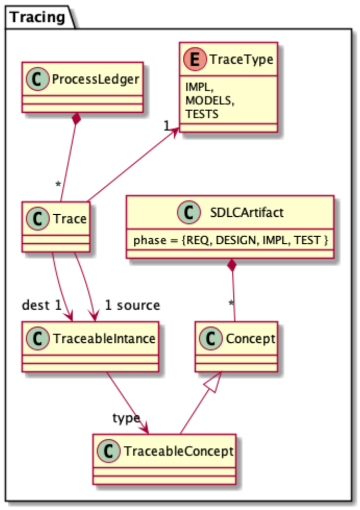
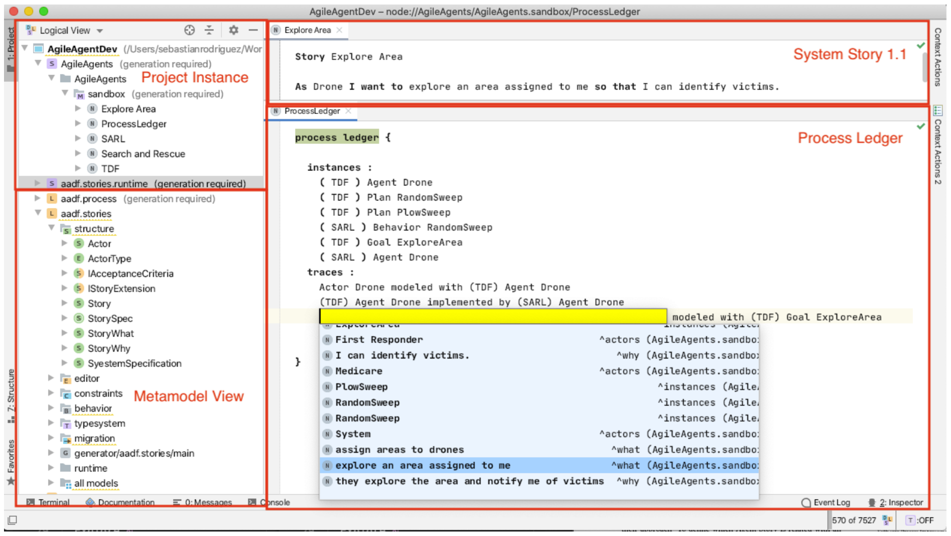


Figure 2: Traceability Framework metamodel



```
Story Explore Area

As Drone I want to explore an area assigned to me so that I can identify victims.
```

```
process ledger {

  instances :
    ( TDF ) Agent Drone
    ( TDF ) Plan RandomSweep
    ( TDF ) Plan PlowSweep
    ( SARL ) Behavior RandomSweep
    ( TDF ) Goal ExploreArea
    ( SARL ) Agent Drone
  traces :
    Actor Drone modeled with (TDF) Agent Drone
    (TDF) Agent Drone implemented by (SARL) Agent Drone
                                  modeled with (TDF) Goal ExploreArea

}
```

```
First Responder                                ^actors (AgileAgents.sandbo
I can identify victims.                           ^why (AgileAgents.sandbo
Medicare                                       ^actors (AgileAgents.sandbo
PlowSweep                                          ^instances (Agile
RandomSweep                                        ^instances (Agile
RandomSweep                                        ^instances (Agile
System                                         ^actors (AgileAgents.sandbo
assign areas to drones                           ^what (AgileAgents.sandbo
explore an area assigned to me                   ^what (AgileAgents.sandbo
they explore the area and notify me of victims    ^why (AgileAgents.sandbo
```

Figure 3: Traceability Support tool

phase. Each SLDCArtifact would contain concepts used in that phase and the specific ones of interest for the purpose of traceability are *TraceableConcepts*. In order words, a TraceableConcept is a concept where we need to keep track of its mappings. For instance, for a TDF design the TraceableConcepts are plans, goals, agents, etc. The particular instances of these concepts used in the system are the *TraceableInstances* that are captured in a particular *Trace*. E.g. the *Plow Sweep* plan or *Explore Area* goal (see Figure 1).

The *Process Ledger* keeps a log of these traces and the relationships. Some example traces from our case study are:

- *Actor* Drone modeled with (TDF) Agent Drone
- *What* explore an area assigned to me modeled with (TDF) Goal ExploreArea
- (TDF) Plan RandomSweep implemented by (SARL) Behavior RandomSweep

Note that the first two traces are related to the mapping of the System Story 1.1 (Table 1) and the last trace is how a plan in the TDF design is realised in the SARL implementation.

Tool support. In order to support the process of capturing the requirements via User and System Stories and maintaining a process ledger of traces for traceability, as described in this Section, we have developed a prototype tool, as tool support is an essential part of any methodology. A screenshot of the development environment can be found in Figure 3. At the bottom-left is the view of the metamodel of our traceability framework, which also encapsulates the metamodel for User and System Stories. Using this metamodel, we can generate our SDLC traceability tool. The top right hand view shows the editor used to define stories with an example (System Story 1.1 from Table 1). At the top-left is the project instance view where the SDLCArtifacts used in the project are defined, such as TDF or SARL. Finally, at the bottom-right is the *Process Ledger* editor. In the process ledger, we define the particular instances of the SDLCArtifacts we are concerned about and then define the traces as can be seen in the figure. The tool provides features such type-safety and auto-completion, as illustrated in the figure in the creation of a trace to track that the *What* of Story 1.1 is *modeled by* the *(TDF) Goal ExploreArea*.

4 EVALUATION

In this section we evaluate our approach by (i) presenting a feature-based comparison with the closest related work; and (ii) conducting a preliminary user study as we detail ahead.

4.1 Feature-based comparison

We are not the first to propose using User Stories in AOSE. Wautelet *et al.* [33] define a process fragment that includes User Stories to gather requirements and creating mappings from the Who/What/Why components to agent concepts. They do not consider acceptance criteria or explicit system-level requirements. They focus on translating User Stories directly into JADEX code [28] without consideration for design artifacts.

The other relevant work is by Carrera *et al.* [5]. They discuss the idea of including *Agent Stories* to represent the view from the "agent's side". They use acceptance criteria (written in the GWT format) of agent stories to create executable agent behaviour tests. However, the process to map user to agent stories remains manual with no guidelines presented. The traceability of these mappings are also unclear.

In order to compare these related work to our approach, we use the following criteria:

User Requirements The approach can capture requirements and functionalities from the user's perspective. Clearly, a crucial motivation for User Stories.

System Requirements The approach can capture system-level requirements. This is important to provide traceability, supporting a path from user/system requirements to code.

Criteria	USS	Wautelet et al.[33]	Carrera et al.[5]
User Requirements	YES	YES	YES
System Requirements	YES	NO	NO (1)
Technology Opportunities	PARTIAL(2)	NO	NO
Ethical Considerations	PARTIAL(2)	NO	NO
AOSE Guidelines	YES	PARTIAL(3)	NO (4)
Traceability Framework	YES	NO	NO
Traceability Tool Support	PARTIAL(5)	NO	NO
Testing Guidelines	PARTIAL(6)	NO	YES
Testing Tool Support	NO	NO	YES

(1) No specific artifact to capture system requirements is proposed. User Stories are translated directly to agent behaviors but lacks details.
(2) We gather options that are traceable concepts in our framework.
(3) User to Agent story is a manual process.
(4) There is a mapping from MAS behaviour to test cases but no explicit mapping from requirement to other AOSE concepts.
(5) Tool is at a prototype stage.
(6) Acceptance criteria is used for test case definition as in traditional development, however no AOSE specific techniques are proposed.

Table 5: Feature based comparison of USS and related work

Technology Opportunties The approach gathers various specialised technological solutions that may need to be considered in the system development stage to realise these requirements. (As discussed in §3.2.)

Ethical Considerations The approach offers mechanisms to capture potential ethical issues when applying technological solutions (also discussed in §3.2).

AOSE Guidelines The approach provides guidelines to translate requirements into AOSE concepts (Design and/or Implementation). Guidelines are important (arguably essential) for supporting developers using the approach.

Traceability Framework The approach provide guidelines on how to trace back artifacts of other SDLC phases to their originating requirements. This is important for maintenance.

Traceability Tool Support The approach includes tooling to support or assist the traceability framework.

Testing Guidelines The approach includes guidelines on how to generate test cases for the AOSE implementation models.

Testing Tool Support The approach includes testing tooling or infrastructure tailored for AOSE concepts.

Table 5 shows our assessment of our approach (User & System Story (USS)), alongside two recent approaches for using User Stories in AOSE, by Wautelet *et al.* [33] and by Carrera *et al.* [5]. This comparison highlights that our approach is richer than the other two approaches in that it provides: (i) System Stories; (ii) (some) support for identifying relevant techniques, and associated ethical issues (iii) detailed guidelines, and (some) support for traceability, including preliminary tool support. On the other hand, the approach of Carrera *et al.* has better support for testing, including tool support.

4.2 Empirical evaluation

Our approach aims to provide an easy-to-use method for agile AOSE, specifically enabling iterative software development. A key feature is the use of System Stories to enhance traceability, which is a difference to prior work. We have shown (§3.3) that our approach provides a mapping to agent design concepts, and argued that our approach supports traceability.

However, providing features to support traceability does not necessarily mean that software developers will find these features effective. More broadly, we also have hypothesised (§1) that our approach is easy to understand, and makes maintenance easier. We therefore conducted a preliminary human participant evaluation with the goal of validating the usefulness of System Stories, and the usability of our overall approach. Our focus was on getting rich qualitative data, that could be analysed to understand what participants were doing, in order to provide insight.

Participants. The scope of our evaluation was limited (hence "preliminary"). We recruited 4 participants, by direct contact. Participants were involved (in varying capacities) in a broader industry project that also had involvement from some of the authors of this paper. The participants were diverse in terms of their relevant background. Participants P1 and P4 work for a government agency, whereas P2 and P3 work for a university in a research support capacity. Although all participants had software engineering experience, only P2 and P3 had expertise in agent-oriented design. Specifically, they both have expertise in TDF, which P1 and P4 did not have. However, only P2 and P4 had experience with agent-oriented programming, and none of the participants had experience or expertise in SARL. Overall, we would characterise P1 and P3 as being novices and P2 and P4 as being experts in developing agent systems.

Participants were briefed verbally, and given a brief explanation of our approach. They were then provided[4] with a one paragraph description of the search and rescue scenario, and then were asked to complete three tasks.

Tasks. The first task (**T1**) asked participants to consider the functionality of the system relating to *searching an area in order to find victims who needed assistance*. Participants were asked to examine a TDF design and indicate which design entities related to this feature (**T1.1**), and then to examine SARL code, and indicate which code entities related to this feature (**T1.2**). Finally, participants were provided with three short videos[5], and were asked to indicate which behaviour(s), visible in the video, related to this feature, and to give a brief description of the behaviour, and why it is related to this feature (**T1.3**). Participants P3 and P4 were given both user and System Stories for this task, whereas P1 and P2 were just given User Stories (see Table 6).

The second task (**T2**) asked participants to do the same three things (identify related parts in a TDF design **T2.1**, in SARL code **T2.2**, and in observed behaviour in videos **T2.3**) for the feature of *guiding a first responder to a victim who requires assistance*. Participants P1 and P2 were given both user and System Stories for this task, whereas P3 and P4 were just given User Stories. T1 and T2 aimed to assess the effect of providing System Stories in addition to User Stories.

By contrast, the third task (**T3**) assessed the usability of our approach. Participants were provided with a one paragraph description of a new feature: *being able to deliver supplies to a first*

[4]Materials at https://github.com/srodriguez/aamas2021-searchrescue-simulation
[5]Video 1 showed a systematic (plow) exploration of a high priority area; Video 2 showed a random walk (low priority area); and Video 3 showed a drone signalling a victim's location to a first responder by hovering and moving up and down.

	P1 (novice) & **P2** (expert)	**P3** (novice) & **P4** (expert)
T1	User Stories only	User & System Stories
T2	User & System Stories	User Stories only
T3	Not applicable - participants wrote stories	

Table 6: Summary of Tasks

responder after they have reached a victim. They were then asked to: **T3.1** identify User Stories for the new feature, **T3.2** derive System Stories for this feature, **T3.3** indicate what they would add/modify in the TDF design to add this feature; and **T3.4** indicate what they would add/modify in the SARL code to add the feature.

Finally, after completing the three tasks, participant completed a short survey that included questions such as "To what extent did you find the requirements for Task 1 comprehensible?" (response scale: 1(very) to 5(incomprehensible)). The survey (see supplementary material) also included a request for any other comments.

At various points along the way participants were asked to note the current time, so we could analyse how long different tasks took to perform. Participants were also instructed to note when they took a break (indicating the start and end time of the break).

Measurements. We analysed what the participants did, i.e. their responses (including, for T3, their User Stories, System Stories, and outline of design and implementation). This included assessing the extent to which participants were able to correctly identify relevant parts of the design, code, and behaviour (T1 & T2) and how long they took to do so; the extent to which participants were able to identify appropriate parts of the design and code to change when adding a new feature (T3) and how long they took to do so; and the participants' responses to the survey, both ordinal and free text.

Results: System Stories. We begin by considering the extent to which it helped to have System Stories provided as well as User Stories. There are a few places where there was clear benefit demonstrated. Firstly, for task T1.3, participant P1 (who only had a User Story) incorrectly described the behaviour seen in the second video. The video shows (random walk) exploration, but P1 indicated it was "*related to Assit* [sic] *Victim plan - I think this is where the drone is guiding the first responders because it seems to head back to the same area. But it isn't clear*". Participants P3 and P4, who had System Stories as well as User Stories, identified the relevant behaviours (Videos 1 & 2) correctly. P2 also identified the videos correctly: we believe that they made use of the TDF diagram to help them (given their expertise in TDF). Secondly, for task T2.3, we see again that having the System Stories helped: the behaviour in Video 3 was correctly described by P1 and P2 (who had System Stories). However, P3 incorrectly indicated it was unrelated, and P4 was correct but uncertain ("*V3 could be drone physically indicating location of victim*", also noting in their survey response "*Only guessing drone behaviour*"). Finally, the survey: P2 indicated that they found T1 (where they did not have System Stories) not really comprehensible (option 4 on the 5 point Likert scale), and that they found identifying relevant parts of the SARL code not really easy to do (also option 4). However, P2 found T2 (where they did have System Stories provided) easy, answering option 2 on the equivalent questions. Similarly, P4 indicated that for T2 (where they did not have System Stories) it was not really easy to identify SARL code, and that identifying the video behaviour for this task was also not really easy (both options 4). By contrast, P4 indicated that for the equivalent questions for T1, it was easier (option 3, neutral). In other words, both P2 and P4, who are experts, found tasks (T1 for P4, and T2 for P2) easier when System Stories were provided.

Results: Usability. Assessed by T3. We observe that all the participants were broadly able to follow the approach, defining User Stories, System Stories, and then going on to outline reasonable changes to the TDF design, and to the code. The usability of our approach was also supported by the survey, where P1, P2 and P3 all indicated that they found the steps somewhat easy to perform (option 2 on a 5-point Likert scale), although P4 was less positive (option 3, neutral). One interesting observation is that the best collection of User Stories was developed by P1, who was a novice in agent-based development (but experienced in non-agent software engineering). This suggests that our approach is able to be used by developers who are not experts in agent-based development.

The analysis of how long participants took did not reveal any useful information: there was too little variance. The longest time taken to perform a single sub-task was 14 minutes (P2 T1.2), but for 8 of the 10 sub-tasks the time taken was less than 10 minutes each.

Finally, P4 commented in the survey that task 2 (specifically identifying relevant SARL code) "*Would had* [sic] *very difficult without seeing TDF diagram first*". This comment highlights the value of having appropriate design models. Although agile is sometimes thought of as eschewing design models in favour of code, in fact the Agile Manifesto[2] "[values] *working software over comprehensive documentation ... That is, while there is value in the items on the right, we value the items on the left more*". In other words, there can be value in having appropriate design artifacts.

5 CONCLUSION

The popularity of agile software methodologies have inspired a number of efforts to bring agile frameworks into AOSE. We can classify these into two categories: those that attempt to modify AOSE to adapt agile concepts (e.g. [7, 14, 22, 31]); and those that adopt agile concepts complementing AOSE (e.g [5, 33]). The approach we present is in the latter category - given an existing system, say implemented in SARL with TDF design artifacts, our approach could be used to elicitate, implement and trace requirements in an iterative agile manner.

Our approach uses User Stories, and introduces the novel concept of System Stories as a "bridge" between User Stories and agent-oriented design artifacts. The approach also supports traceability of requirements. Our preliminary user evaluation has shown that it is usable, and that System Stories do provide benefit. Our approach is richer than prior work not just in the inclusion of System Stories, but also in providing guidelines and support for traceability (including tool support). While richer, our approach is still light-weight, providing a simple and rapid path from requirements to code.

There are a few areas for future work. Our approach includes identifying relevant technologies, and assessing associated ethical issues. However, this aspect of the approach has not been developed in detail, nor evaluated. There is also scope for improving our support for deriving tests, including looking at tool support. Finally, future work could include a larger scale evaluation.

REFERENCES

[1] Kent Beck and Cynthia Andres. 2005. *Extreme Programming Explained: Embrace Change* (2nd ed ed.). Addison-Wesley, Boston, MA.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mallor, Ken Shwaber, and Jeff Sutherland. 2001. *The Agile Manifesto*. Technical Report. The Agile Alliance.

[3] R. H. Bordini, J. F. Hubner, and M. Wooldridge. 2007. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley and Sons.

[4] Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. 2004. TROPOS: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems* 8 (May 2004), 203–236.

[5] Álvaro Carrera, Carlos A. Iglesias, and Mercedes Garijo. 2014. Beast Methodology: An Agile Testing Methodology for Multi-Agent Systems Based on Behaviour Driven Development. *Information Systems Frontiers* 16, 2 (April 2014), 169–182. https://doi.org/10.1007/s10796-013-9438-5

[6] Antonio Chella, Massimo Cossentino, and Luca Sabatucci. 2004. Tools and patterns in designing multi-agent systems with PASSI. *WSEAS Transactions on Communications* 3, 1 (2004), 352–358.

[7] Antonio Chella, Massimo Cossentino, Luca Sabatucci, and Valeria Seidita. 2006. Agile PASSI: An Agile Process for Designing Agents. *Comput. Syst. Sci. Eng.* (2006).

[8] Mike Cohn. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley, Boston.

[9] Mike Cohn. 2010. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley, Upper Saddle River, NJ.

[10] Massimo Cossentino, Vincent Hilaire, Nicolas Gaud, Stéphane Galland, and Abderrafiaa Koukam. 2014. The ASPECS Process. In *Handbook on Agent-Oriented Design Processes*, Massimo Cossentino, Vincent Hilaire, Andrea Molesini, and Valeria Seidita (Eds.). Springer, 65–114.

[11] Scott A. DeLoach and Juan C. García-Ojeda. 2010. O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems. *Int. J. Agent Oriented Softw. Eng.* 4, 3 (2010), 244–280. https://doi.org/10.1504/IJAOSE.2010.036984

[12] Scott A. DeLoach, Lin Padgham, Anna Perini, Angelo Susi, and John Thangarajah. 2009. Using three AOSE toolkits to develop a sample design. *International Journal of Agent-Oriented Software Engineering* 3, 4 (2009), 416–476.

[13] Virginia Dignum. 2019. *Responsible Artificial Intelligence - How to Develop and Use AI in a Responsible Way*. Springer. https://doi.org/10.1007/978-3-030-30371-6

[14] Jaschar Domann, Sindy Hartmann, Michael Burkhardt, Alexander Barge, and Sahin Albayrak. 2014. An Agile Method for Multiagent Software Engineering. *Procedia Computer Science* 32 (Jan. 2014), 928–934. https://doi.org/10.1016/j.procs.2014.05.513

[15] Rick Evertsz and John Thangarajah. 2020. A Framework for Engineering Human/Agent Teaming Systems. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2477–2484.

[16] Rick Evertsz, John Thangarajah, and Thanh Ly. 2019. *Practical Modelling of Dynamic Decision Making*. Springer. https://doi.org/10.1007/978-3-319-95195-9

[17] Rick Evertsz, John Thangarajah, Thanh Ly, and Nitin Yadav. 2015. Agent Oriented Modelling of Tactical Decision Making. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2015)*. Istanbul, Turkey, 1051–1060.

[18] Rick Evertsz, John Thangarajah, and Michael Papasimeon. 2017. The Conceptual Modelling of Dynamic Teams for Autonomous Systems. In *Conceptual Modeling - 36th International Conference, ER 2017, Valencia, Spain, November 6-9, 2017, Proceedings*. 311–324.

[19] M. Fowler. 2003. Who Needs an Architect? *IEEE Software* 20, 5 (Sept. 2003), 11–13. https://doi.org/10.1109/MS.2003.1231144

[20] Martin Fowler. 2007. Design Stamina Hypothesis. https://martinfowler.com/bliki/DesignStaminaHypothesis.html.

[21] Alma María Gómez-Rodríguez and Juan Carlos González Moreno. 2010. Comparing Agile Processes for Agent Oriented Software Engineering. In *Product-Focused Software Process Improvement, 11th International Conference, PROFES 2010, Limerick, Ireland, June 21-23, 2010. Proceedings (Lecture Notes in Business Information Processing)*, Muhammad Ali Babar, Matias Vierimaa, and Markku Oivo (Eds.), Vol. 6156. Springer, 206–219. https://doi.org/10.1007/978-3-642-13792-1_17

[22] Juan C. González-Moreno, Alma Gómez-Rodríguez, Rubén Fuentes-Fernández, and David Ramos-Valcárcel. 2014. INGENIAS-Scrum. In *Handbook on Agent-Oriented Design Processes*, Massimo Cossentino, Vincent Hilaire, Ambra Molesini, and Valeria Seidita (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 219–251. https://doi.org/10.1007/978-3-642-39975-6_8

[23] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and Patrick Mäder. 2012. Traceability Fundamentals. In *Software and Systems Traceability*, Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman (Eds.). Springer, London, 3–22. https://doi.org/10.1007/978-1-4471-2239-5_1

[24] Dan North. 2006. Introducing BDD. https://dannorth.net/introducing-bdd/.

[25] Lin Padgham and Michael Winikoff. 2004. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons.

[26] Juan Pavón, Jorge J Gómez-Sanz, and Rubén Fuentes. 2005. The INGENIAS methodology and tools. *Agent-oriented methodologies* 9 (2005), 236–276.

[27] PMI. 2017. *Pulse of the Profession 2017*. Technical Report. Project Management Institute.

[28] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. 2005. Jadex: A BDI Reasoning Engine. In *Multi-Agent Programming*. Springer, 149–174.

[29] Sebastian Rodriguez, Nicolas Gaud, and Stéphane Galland. 2014. SARL: A General-Purpose Agent-Oriented Programming Language. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Warsaw, Poland, August 11-14, 2014 - Volume III*. IEEE Computer Society, 103–110. https://doi.org/10.1109/WI-IAT.2014.156

[30] Ken Schwaber and Jeff Sutherland. 2017. *The Scrum Guide*. Technical Report. 20 pages.

[31] Jan-Philipp Steghöfer, Hella Seebach, Benedikt Eberhardinger, Michael Huebschmann, and Wolfgang Reif. 2015. Combining PosoMAS Method Content with Scrum: Agile Software Engineering for Open Self-Organising Systems. *Scalable Comput. Pract. Exp.* 16, 4 (2015), 333–354. http://www.scpe.org/index.php/scpe/article/view/1127

[32] The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems. 2017. Ethically Aligned Design: A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems, Version 2. http://standards.ieee.org/develop/indconn/ec/autonomous_systems.html.

[33] Yves Wautelet, Samedi Heng, Soreangsey Kiv, and Manuel Kolp. 2017. User-Story Driven Development of Multi-Agent Systems: A Process Fragment for Agile Methods. *Computer Languages, Systems & Structures* 50 (Dec. 2017), 159–176. https://doi.org/10.1016/j.cl.2017.06.007

[34] Michael Winikoff. 2005. JACK Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming*. Springer, New York, NY, 175–193.

[35] Michael Wooldridge, Nicholas R Jennings, and David Kinny. 2000. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and multi-agent systems* 3, 3 (2000), 285–312.