Agile Approach for Agent Oriented Software Engineering

Sebastian Rodriguez

sebastian.rodriguez@rmit.edu.au

Work in Collaboration with John Thangarajah and Michael Winikoff

RMIT University – CIAIRI

EMAS Workshop @ AAMAS’24

7 May 2024 - Auckland, NZ

RMIT UNIVERSITY

Search and Rescue Scenario

Drones assist in locating and identifying victims, via tasks assigned to them by the human drone operator which they carry out autonomously.

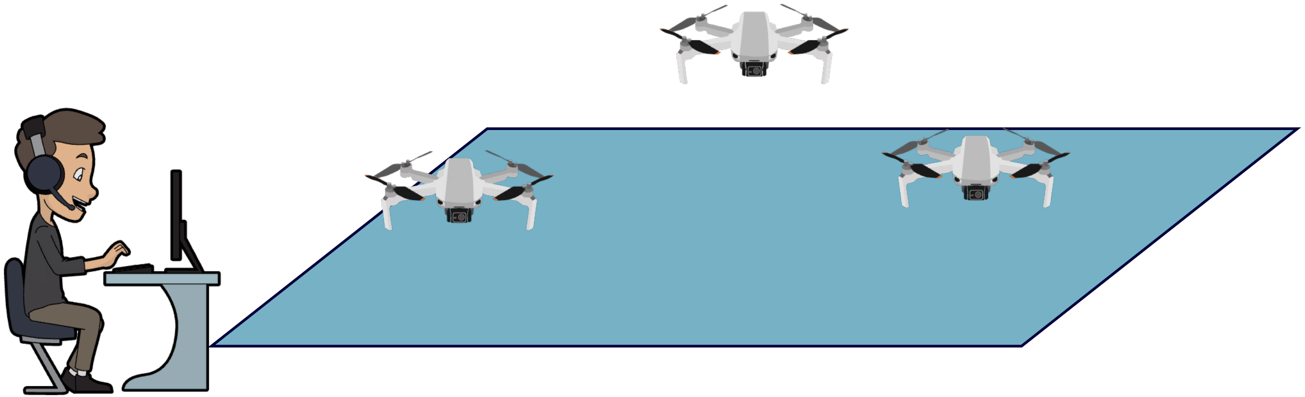
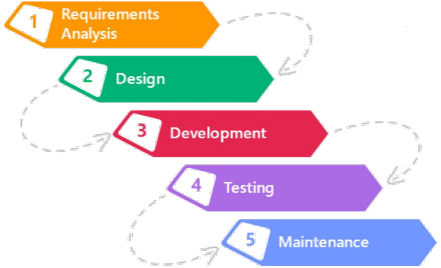


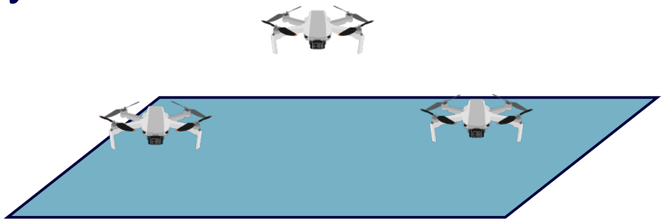
Image: wikimedia

Search and Rescue Scenario

Drones assist in locating and identifying victims, via tasks assigned to them by the human drone operator which they carry out autonomously.

Agile AOSE Software Development Lifecycle?





Requirements:
- Autonomous Exploration
- Victim detection
- Human-Machine Interaction
- Explainability (!)

Agent Models and Programming

Testing, Evaluation, Verification and Validation

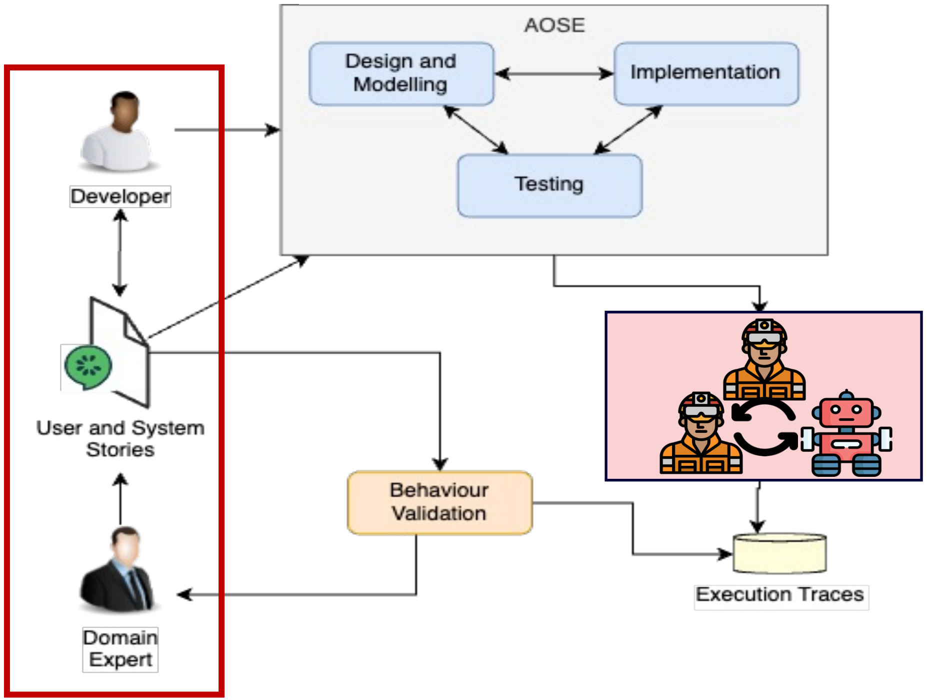
Runtime

Image: wikimedia

Agent-oriented Software Engineering approach

Agent Models and Programming

Requirements



Runtime

Testing, Evaluation, Verification and Validation

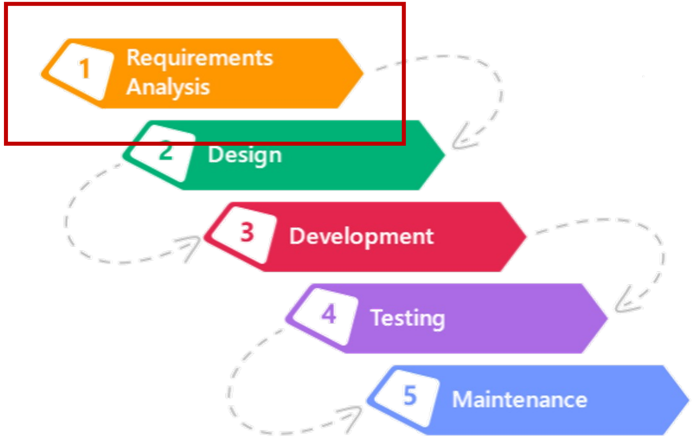
Requirements
Agile and User Stories

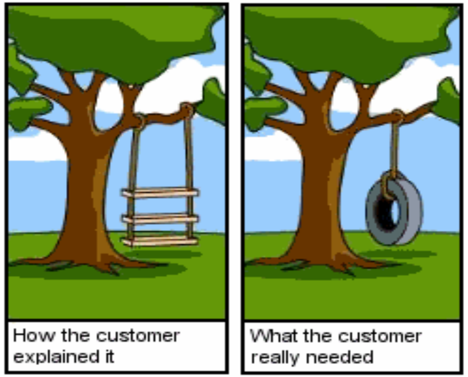
Requirement elicitation and gathering is critical in Software Development

Agile is widely used and accepted in the SE industry

User Stories are commonly accepted by agile practitioners

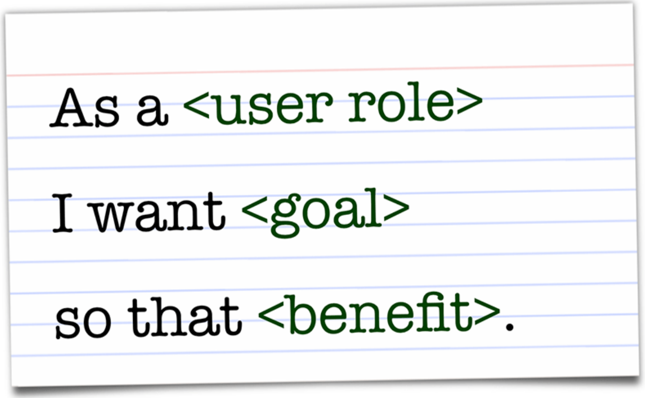
If we don't have good requirements, we are not going to build the right system.



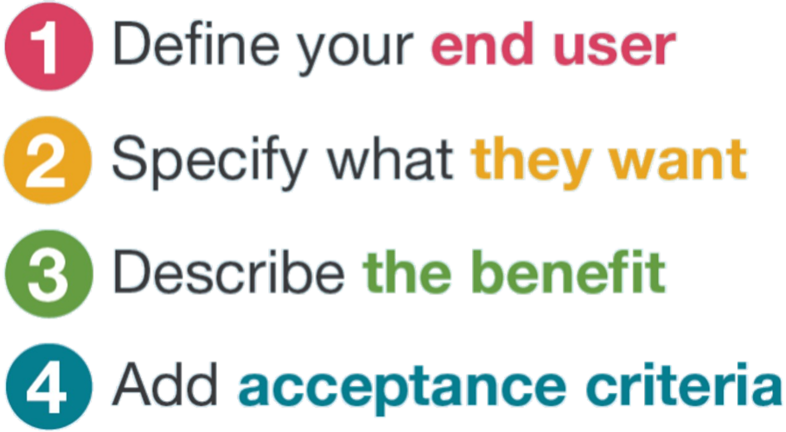

How the customer explained it
What the customer really needed

User Stories

A **user story** is an informal, natural language description of one or more features of a software system. **User stories** are often written from the **perspective of an end user or user of a system**.



As a <user role>
I want <goal>
so that <benefit>.



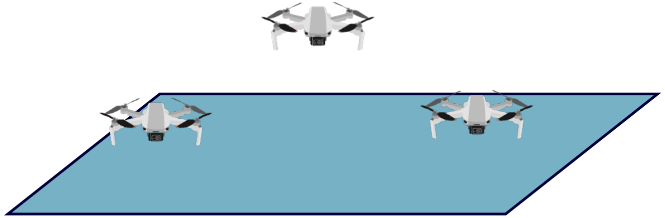
1. Define your **end user**
2. Specify what **they want**
3. Describe **the benefit**
4. Add **acceptance criteria**

Intelligent Autonomous Systems Requirements

As Drone Operator,

I want drones to explore autonomously a given area

So that they find victims and notify me

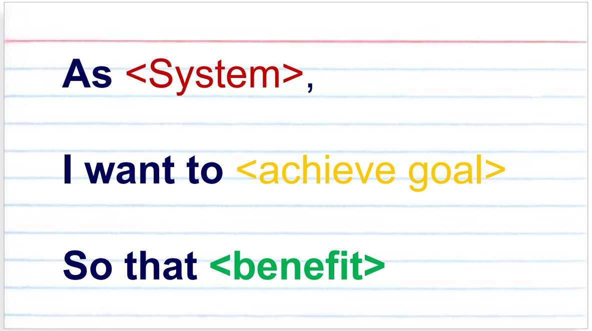


Problems:

1) User Story can be too large to fit in ONE iteration
2) Difficult to split in stories from the ”End user perspective”

System Stories: Idea

A **system story** is an informal, natural language description of one **feature** of the system **from the system's perspective required to fulfill one or more user stories**



As <System>,

I want to <achieve goal>

So that **<benefit>**

Benefits

- ✓ Clear link between User and System-level requirements
- ✓ Consider the system as a first-class citizen

USS Approach

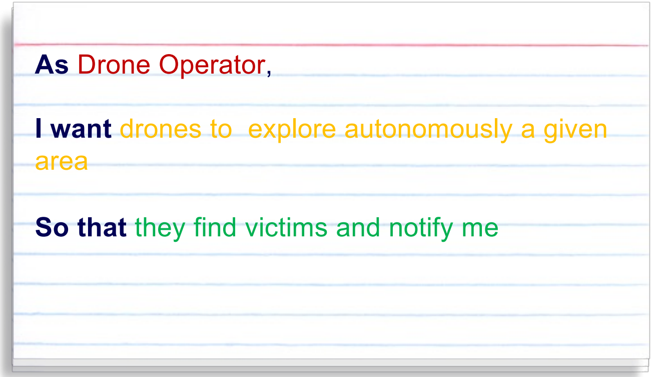
Given a high-level specification of the system in terms of objectives:

(1) **identify User Stories** using classical techniques

(2) refine into **System Stories and their acceptance criteria**; and

(3) during the development process:
map the System Stories to the relevant agent concepts.
maintain a process ledger for the purpose of **traceability**

Refine each User Story into System Stories

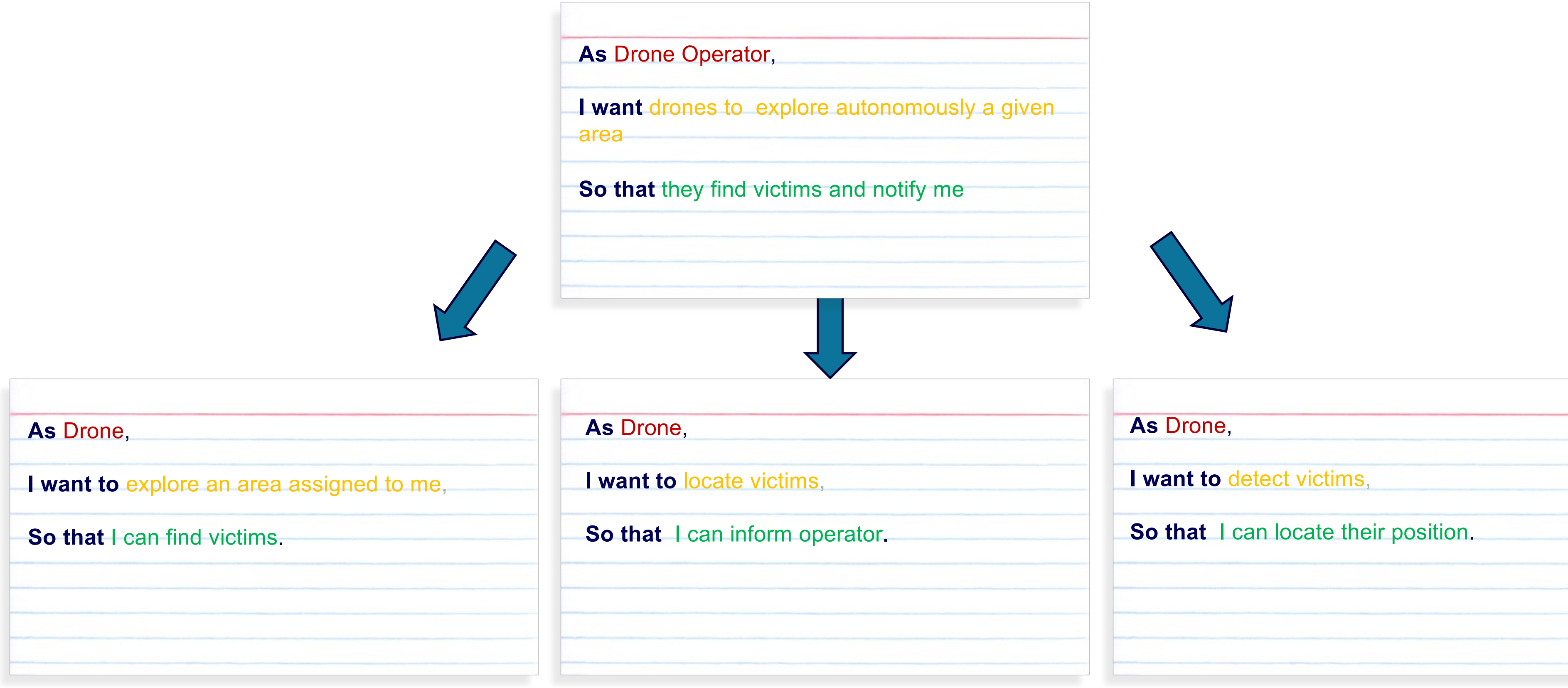


As Drone Operator,

I want drones to explore autonomously a given area

So that they find victims and notify me

Refine each User Story into System Stories



As Drone Operator,

I want drones to explore autonomously a given area

So that they find victims and notify me

As Drone,

I want to explore an area assigned to me,

So that I can find victims.

As Drone,

I want to locate victims,

So that I can inform operator.

As Drone,

I want to detect victims,

So that I can locate their position.

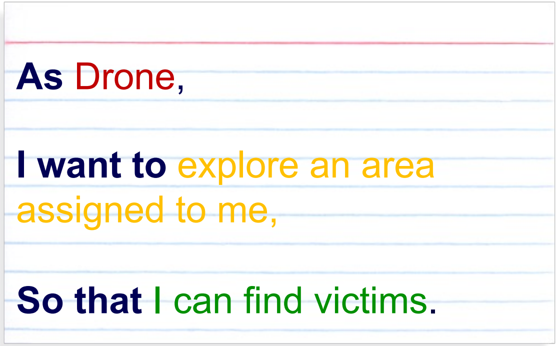
Requirements

Capture system requirements using User and System Stories

User Story

1. Define your **end user**
2. Specify what **they want**
3. Describe **the benefit**
4. Add **acceptance criteria**

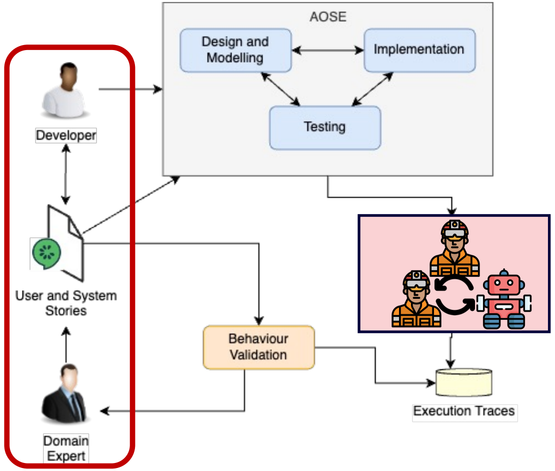
System Story



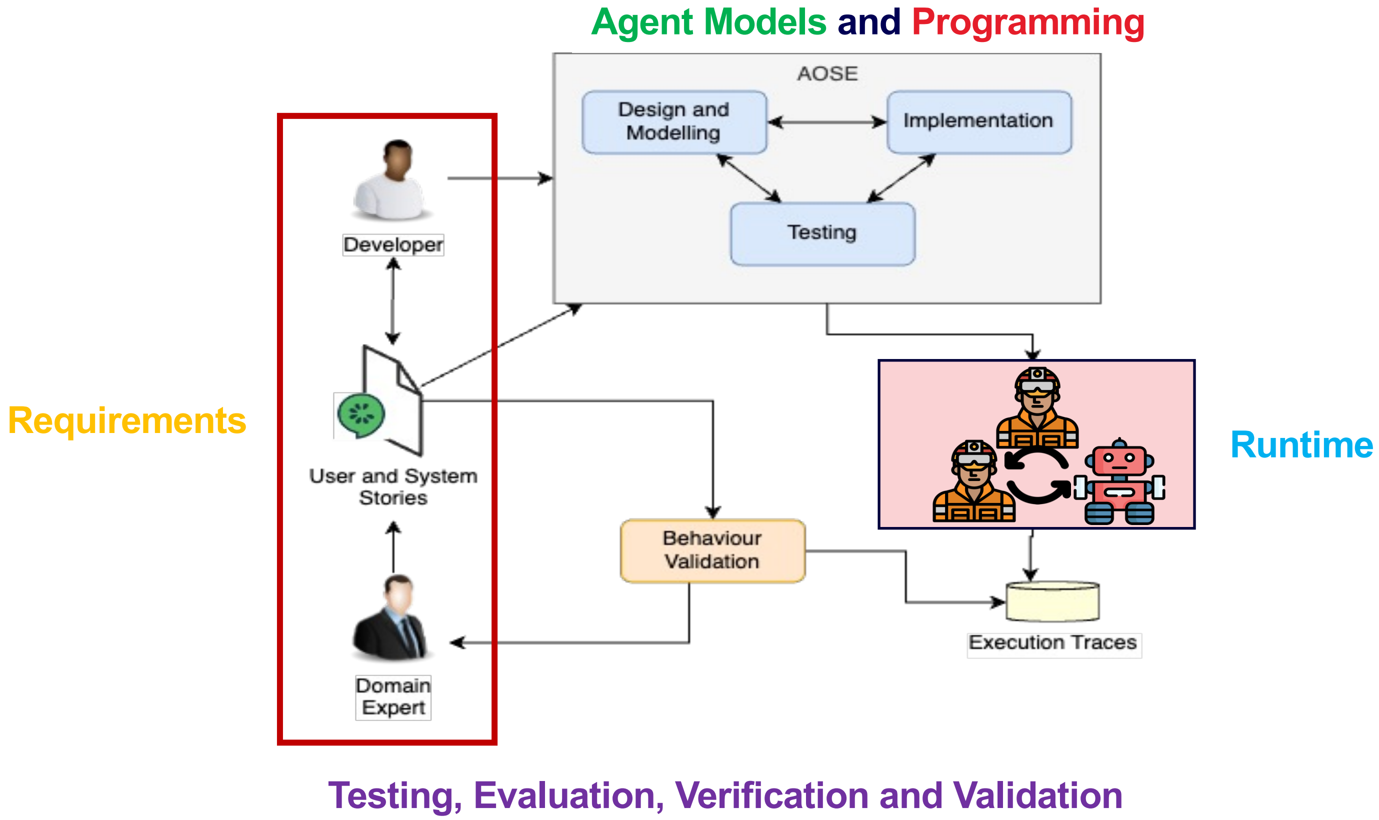
As Drone,

I want to explore an area assigned to me,

So that I can find victims.



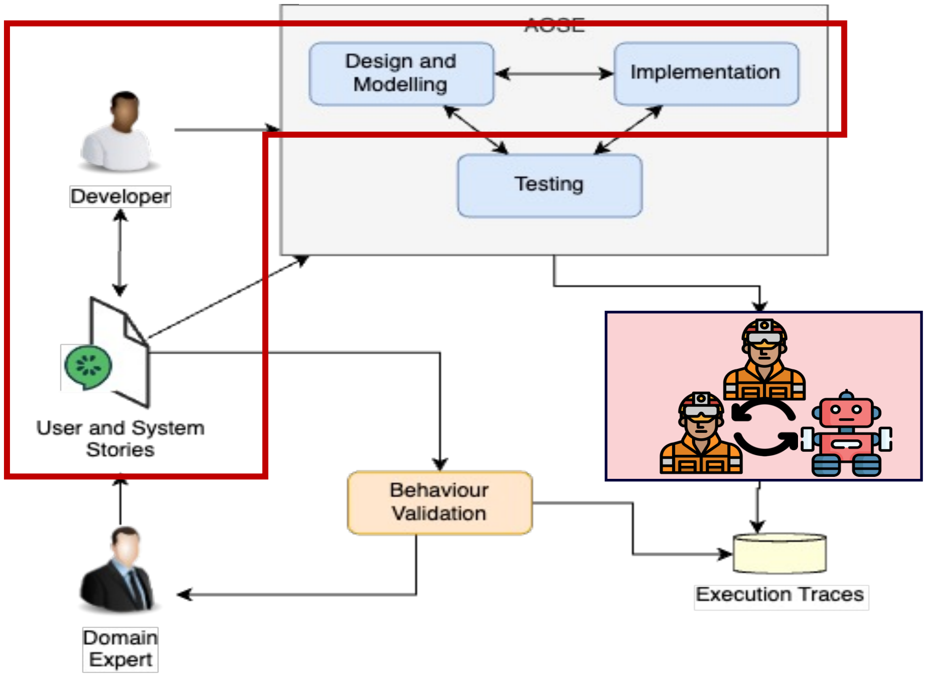
Agent-oriented Software Engineering approach



Agent-oriented Software Engineering approach

Agent Models and Programming

Requirements

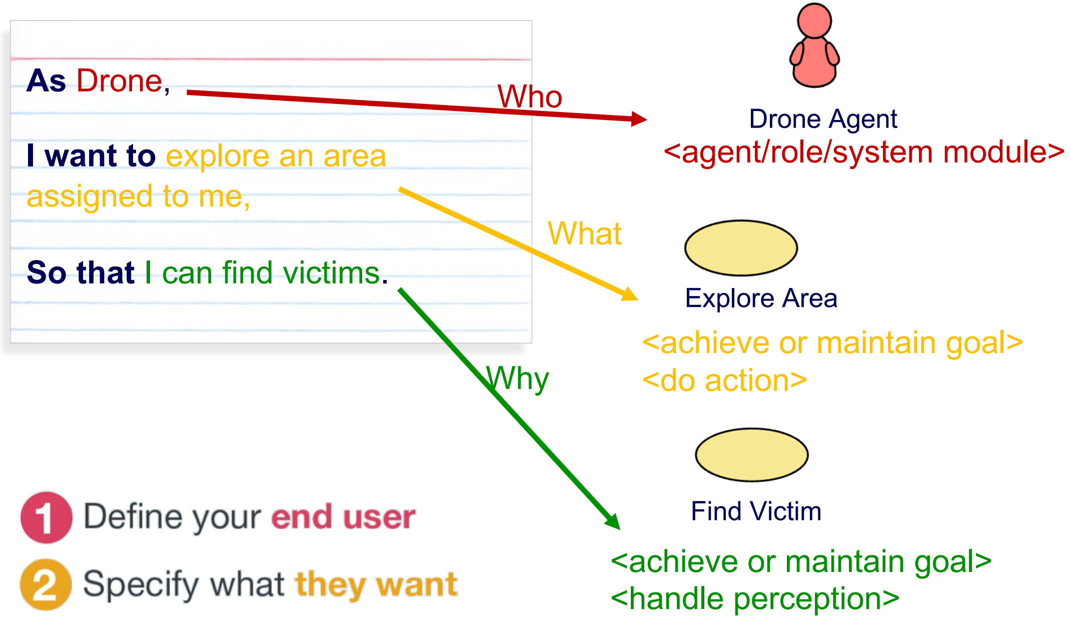


Runtime

Testing, Evaluation, Verification and Validation

Requirements to Agents (AAMAS'21)

Requirements → Models → Implementation



As Drone,

I want to explore an area assigned to me,

So that I can find victims.

Who

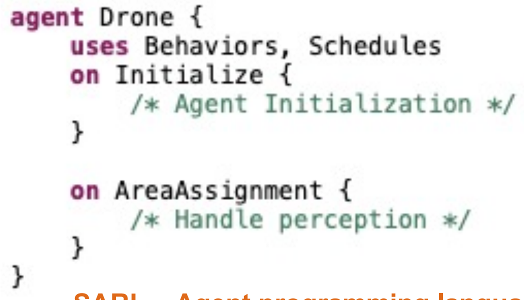
What

Why

Drone Agent
<agent/role/system module>

Explore Area
<achieve or maintain goal>
<do action>

Find Victim
<achieve or maintain goal>
<handle perception>



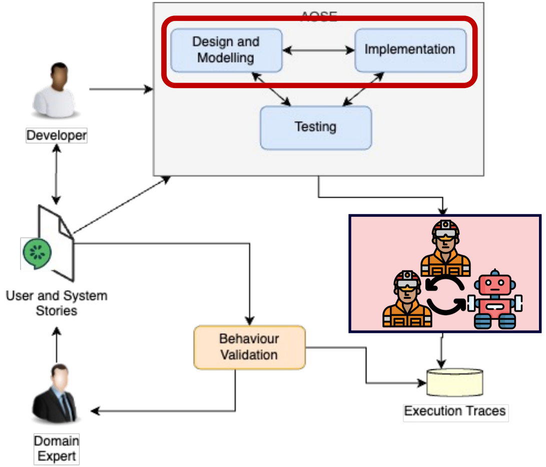
```
agent Drone {
    uses Behaviors, Schedules
    on Initialize {
        /* Agent Initialization */
    }

    on AreaAssignment {
        /* Handle perception */
    }
}
```

SARL – Agent programming language
http://www.sarl.io

1. Define your **end user**
2. Specify what **they want**
3. Describe **the benefit**
4. Add **acceptance criteria**

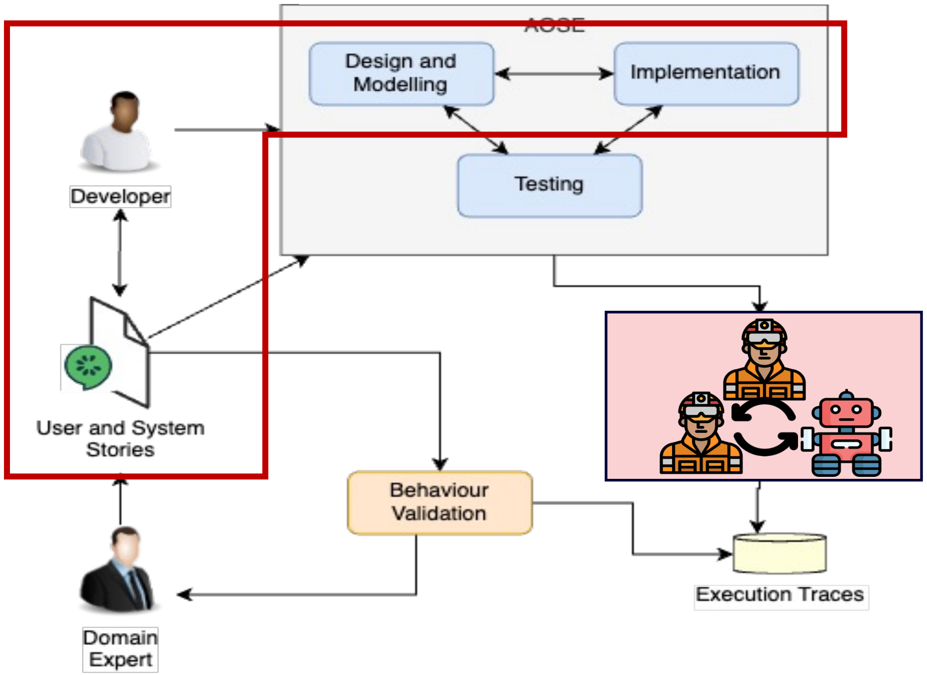
Tactical Development Framework

http://www.agentprojects.com/tdf/ TDF



Agent-oriented Software Engineering approach

Agent Models and Programming



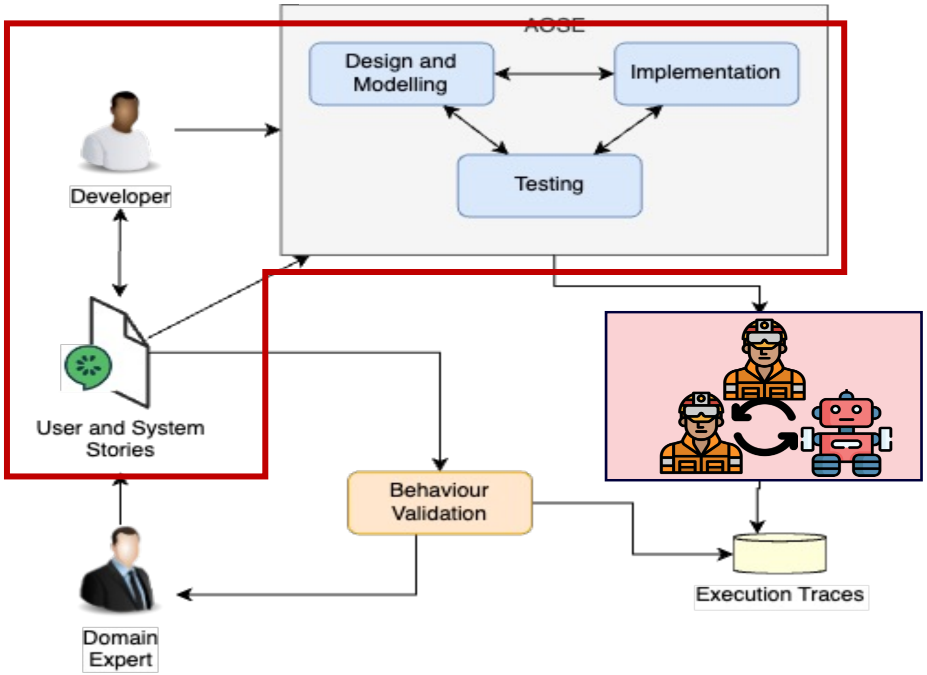
Requirements

Runtime

Testing, Evaluation, Verification and Validation

Agent-oriented Software Engineering approach

Agent Models and Programming



Requirements

Runtime

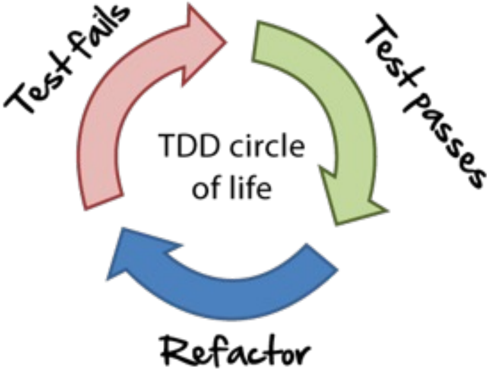
Testing, Evaluation, Verification and Validation

Testing, Evaluation, Verification and Validation

AAMAS’23

Objective

- **Adopt Test-Driven approach for Agent development**
- Verify individual agent behaviors against requirements
- Verify System behavior against requirements



Constraints

- Integrate with traditional SE tools and techniques
- Facilitate requirements validation with SMEs
- Validate of test suite quality

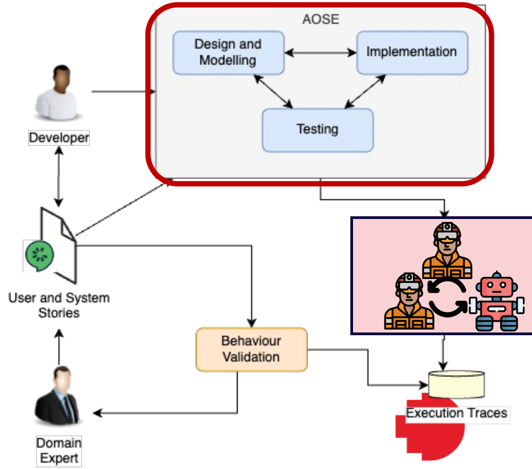
Behaviour driven approach for agent system

1. Define your **end user**
2. Specify what **they want**
3. Describe **the benefit**
4. Add **acceptance criteria**

Acceptance Criteria:
conditions that a software product must meet to be accepted by a user, a customer, or other system.

ACCEPTED

REJECTED



Extensions to USS for BDD

Adopt Scenario based Acceptance Criteria

- Originated with BDD

Define types of System Stories

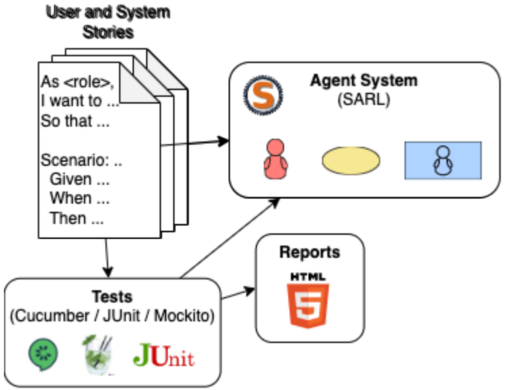
- Goal
- Plan
- Belief
- Perception

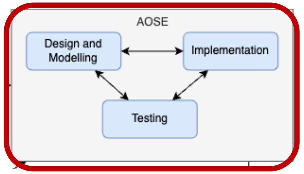
Define Guidelines to capture acceptance criteria

Integrated with Industry-grade testing tools

- Gherkin Language
- Cucumber
- Junit

Scenario: <title>
Given <state>
When <trigger>
Then <expected outcome>



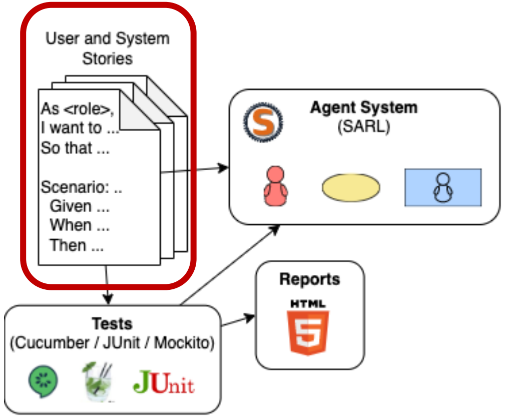


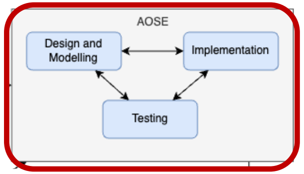
Extensions to USS for BDD

Goal Story Example

@goal
Feature: Explore Area
As Drone,
I want to explore areas assigned to me
So that I can find victims

@goal-success
…
@goal-failure
…
@goal-context
…
@goal-plan
…

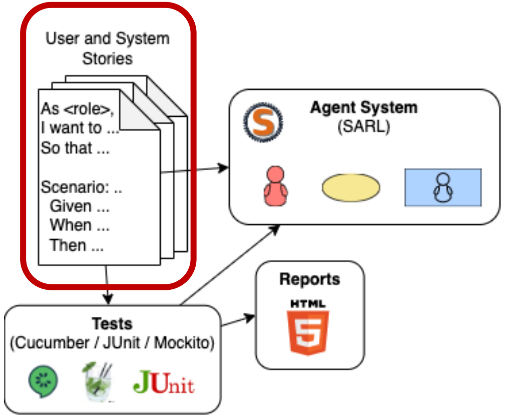


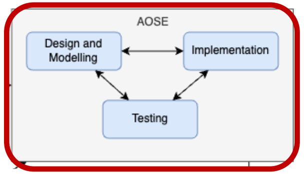


Extensions to USS for BDD

Goal Story Example

```
@goal
Feature: Explore Area
  As Drone,
  I want to explore areas assigned to me
  So that I can find victims

 @goal-success
 Scenario: Goal success
   Given I believe current_area_explored is greater than 95%
   When I evaluate current_goal success
   Then goal success is true
 @goal-failure
 …
 @goal-context
 …
 @goal-plan
 …
```

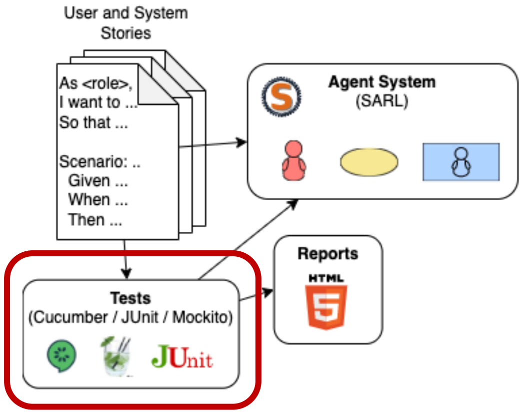


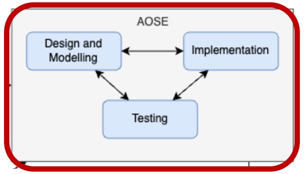


Extensions to USS for BDD

```
...
@goal-success
Scenario: Goal success
  Given I believe current_area_explored is greater than 95%
  When I evaluate current_goal success
  Then goal success is true
```

```
class ExploreAreaTestSteps {
...
@Given("I believe current_area_explored is greater than {int}%")
def exploration_is_percent(rate : int) {
		val area = new Area(0f, 0f, 10f, 10f, Priority.HIGH)
		doReturn(area).when(this.agt.beliefs).currentArea
		doReturn(rate /
100f).when(this.agt.beliefs).explorationRate(any(Area))
}
@When("I evaluate current_goal success")
def evaluate_goal_success {
		this.evalResult = this.goal.success
}
@Then("goal {word} is {word}")
def evaluation_outcome(cond : String, outcome : String) {
		assertEquals(Boolean.valueOf(outcome), this.evalResult)
}
}
```



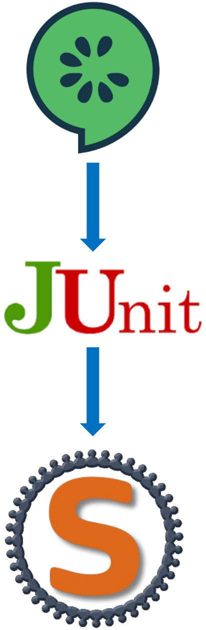


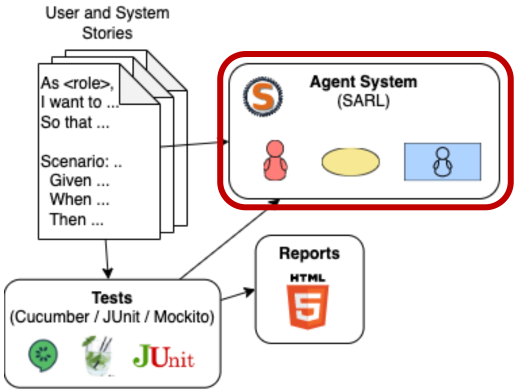
Extensions to USS for BDD

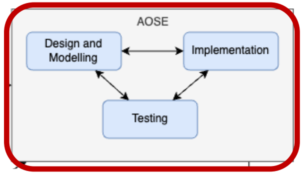
```
...
@goal-success
Scenario: Goal success
   Given I believe current_area_explored is greater than 95%
   When I evaluate current_goal success
   Then goal success is true
```

```
class ExploreAreaTestSteps {
...
@Given("I believe current_area_explored is greater than {int}%")
def exploration_is_percent(rate : int) {…}
@When("I evaluate current_goal success")
def evaluate_goal_success {…}
@Then("goal {word} is {word}")
def evaluation_outcome(cond : String, outcome : String) {…}
}
```

```
skill ExploreArea extends Goal implements AchievementGoal{
  uses SearchRescueBeliefs, DroneState
  def context : boolean {…}
  def success : boolean {
    explorationRate(currentArea) >= 0.95f
  }
  def failure : boolean {…}
}
```





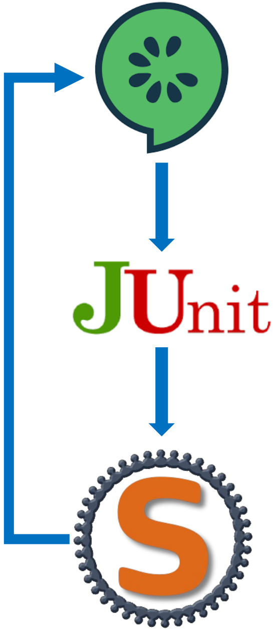


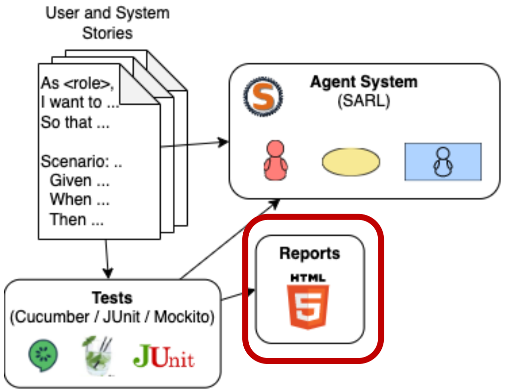
Extensions to USS for BDD

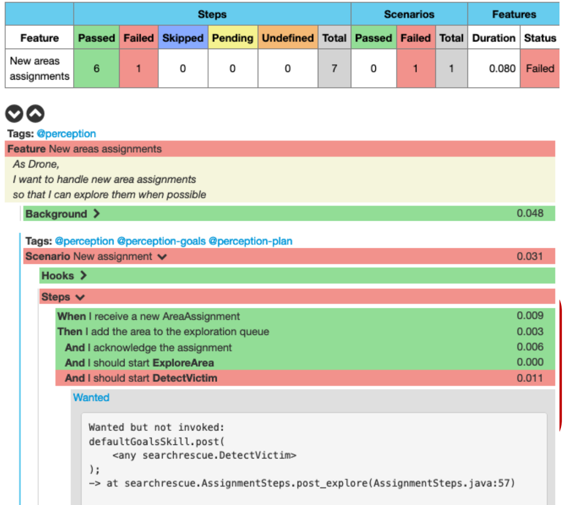
```
…
@goal-success
Scenario: Goal success
   Given I believe current_area_explored is greater than 95%
   When I evaluate current_goal success
   Then goal success is true
```

```
class ExploreAreaTestSteps {
…
@Given("I believe current_area_explored is greater than {int}%")
def exploration_is_percent(rate : int) {…}
@When("I evaluate current_goal success")
def evaluate_goal_success {…}
@Then("goal {word} is {word}")
def evaluation_outcome(cond : String, outcome : String) {…}
}
```

```
skill ExploreArea extends Goal implements AchievementGoal{
  uses SearchRescueBeliefs, DroneState
  def context : boolean {…}
  def success : boolean {
    explorationRate(currentArea) >= 0.95f
  }
  def failure : boolean {…}
}
```







	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
New areas assignments	6	1	0	0	0	7	0	1	1	0.080	Failed

Tags: @perception

Feature New areas assignments

As Drone,
I want to handle new area assignments
so that I can explore them when possible

Background › 0.048

Tags: @perception @perception-goals @perception-plan

Scenario New assignment ⌄ 0.031

Hooks ›

Steps ⌄

When I receive a new AreaAssignment 0.009
Then I add the area to the exploration queue 0.003
And I acknowledge the assignment 0.006
And I should start **ExploreArea** 0.000
And I should start **DetectVictim** 0.011

Wanted

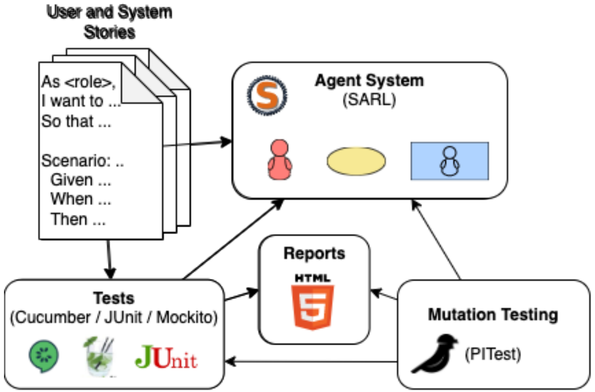
```
Wanted but not invoked:
defaultGoalsSkill.post(
    <any searchrescue.DetectVictim>
);
-> at searchrescue.AssignmentSteps.post_explore(AssignmentSteps.java:57)
```

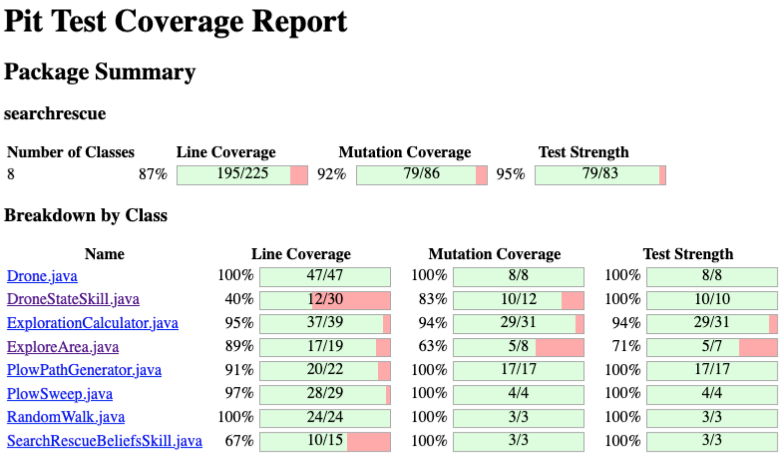
Test Quality Evaluation

Mutation Testing via PI Test

Analysis

- Discovering missing acceptance criteria
- Identifying Ground beliefs
- Acceptable behaviours despite mutation survival





Pit Test Coverage Report

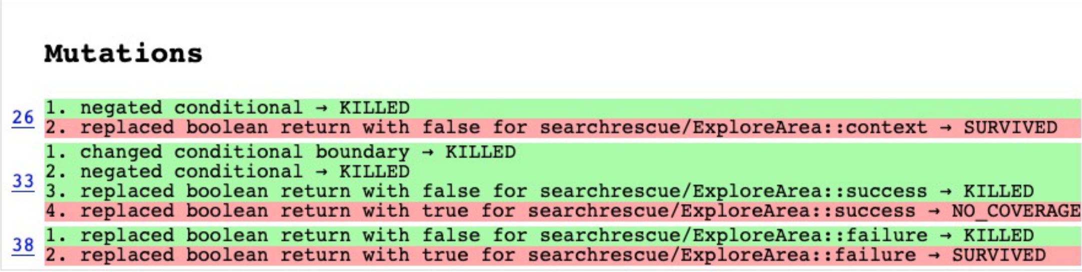
Package Summary

searchrescue

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
8	87%	195/225	92%	79/86	95%	79/83

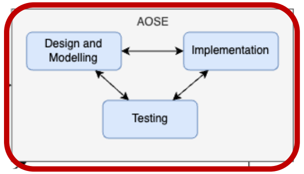
Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Drone.java	100%	47/47	100%	8/8	100%	8/8
DroneStateSkill.java	40%	12/30	83%	10/12	100%	10/10
ExplorationCalculator.java	95%	37/39	94%	29/31	94%	29/31
ExploreArea.java	89%	17/19	63%	5/8	71%	5/7
PlowPathGenerator.java	91%	20/22	100%	17/17	100%	17/17
PlowSweep.java	97%	28/29	100%	4/4	100%	4/4
RandomWalk.java	100%	24/24	100%	3/3	100%	3/3
SearchRescueBeliefsSkill.java	67%	10/15	100%	3/3	100%	3/3



Mutations

```
26  1. negated conditional → KILLED
    2. replaced boolean return with false for searchrescue/ExploreArea::context → SURVIVED
33  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
    3. replaced boolean return with false for searchrescue/ExploreArea::success → KILLED
    4. replaced boolean return with true for searchrescue/ExploreArea::success → NO_COVERAGE
38  1. replaced boolean return with false for searchrescue/ExploreArea::failure → KILLED
    2. replaced boolean return with true for searchrescue/ExploreArea::failure → SURVIVED
```



Tooling integration

Full IDE support (via SARL IDE)

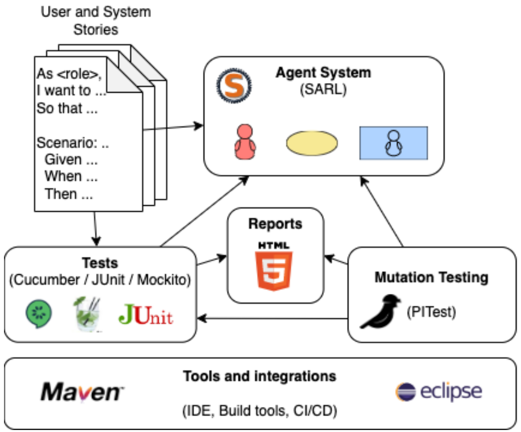
- Debugging with breakpoints
- Code inspection

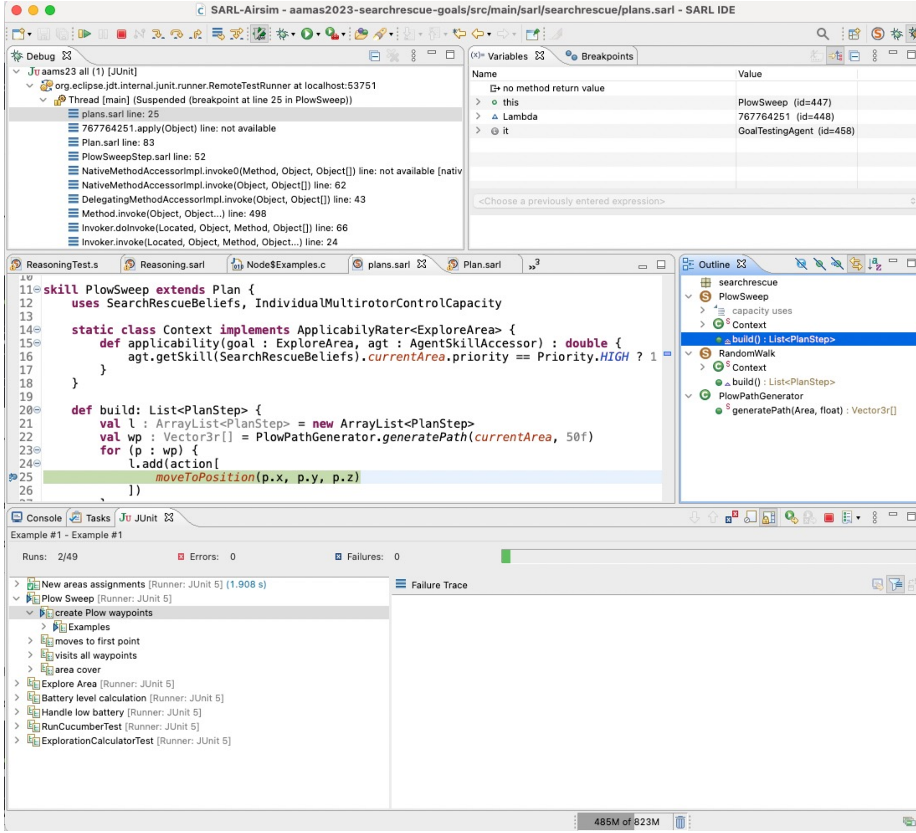
Mainstream Testing frameworks

Tools to verify tests suite quality

Building and Deployment tools

- Enables Continuous Integration and Delivery




SARL-Airsim - aamas2023-searchrescue-goals/src/main/sarl/searchrescue/plans.sarl - SARL IDE

Debug

```
aams23 all (1) [JUnit]
  org.eclipse.jdt.internal.junit.runner.RemoteTestRunner at localhost:53751
    Thread [main] (Suspended (breakpoint at line 25 in PlowSweep))
      plans.sarl line: 25
      767764251.apply(Object) line: not available
      Plan.sarl line: 83
      PlowSweepStep.sarl line: 52
      NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [nativ
      NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
      DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
      Method.invoke(Object, Object...) line: 498
      Invoker.doInvoke(Located, Object, Method, Object[]) line: 66
      Invoker.invoke(Located, Object, Method, Object...) line: 24
```

Variables | Breakpoints

Name	Value
no method return value	
this	PlowSweep (id=447)
Lambda	767764251 (id=448)
it	GoalTestingAgent (id=458)

<Choose a previously entered expression>

ReasoningTest.s | Reasoning.sarl | Node$Examples.c | plans.sarl | Plan.sarl | »3

```
skill PlowSweep extends Plan {
    uses SearchRescueBeliefs, IndividualMultirotorControlCapacity

    static class Context implements ApplicabilyRater<ExploreArea> {
        def applicability(goal : ExploreArea, agt : AgentSkillAccessor) : double {
            agt.getSkill(SearchRescueBeliefs).currentArea.priority == Priority.HIGH ? 1
        }
    }

    def build: List<PlanStep> {
        val l : ArrayList<PlanStep> = new ArrayList<PlanStep>
        val wp : Vector3r[] = PlowPathGenerator.generatePath(currentArea, 50f)
        for (p : wp) {
            l.add(action[
                moveToPosition(p.x, p.y, p.z)
            ])
```

Outline

```
searchrescue
PlowSweep
  capacity uses
  Context
  build() : List<PlanStep>
RandomWalk
  Context
  build() : List<PlanStep>
PlowPathGenerator
  generatePath(Area, float) : Vector3r[]
```

Console | Tasks | JUnit

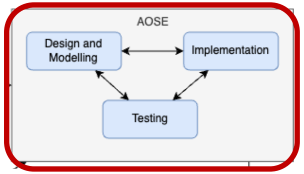
Example #1 - Example #1

Runs: 2/49 Errors: 0 Failures: 0

```
New areas assignments [Runner: JUnit 5] (1.908 s)
Plow Sweep [Runner: JUnit 5]
  create Plow waypoints
    Examples
  moves to first point
  visits all waypoints
  area cover
Explore Area [Runner: JUnit 5]
Battery level calculation [Runner: JUnit 5]
Handle low battery [Runner: JUnit 5]
RunCucumberTest [Runner: JUnit 5]
ExplorationCalculatorTest [Runner: JUnit 5]
```

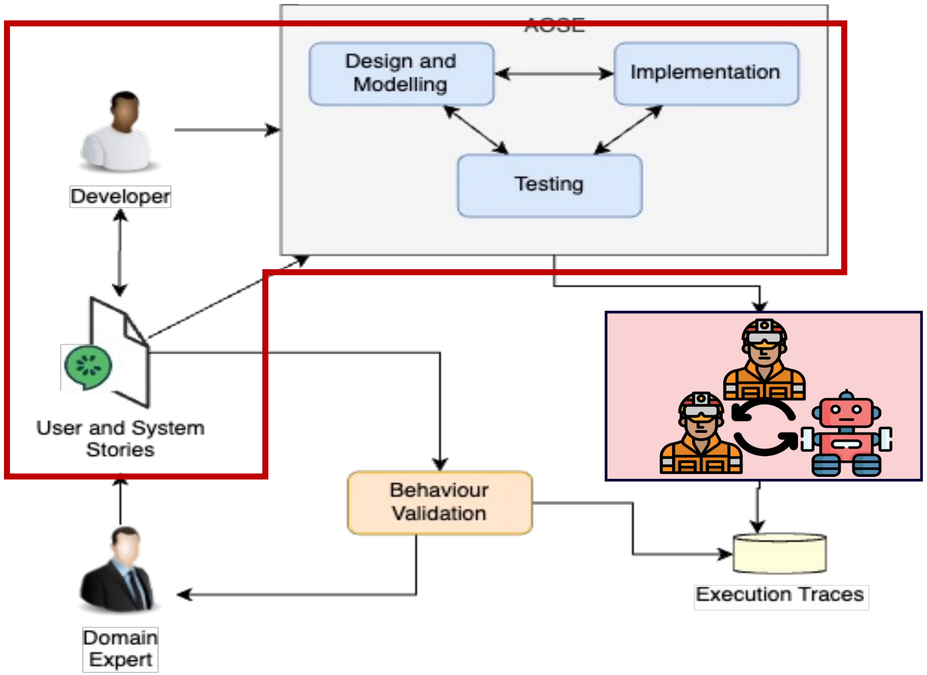
Failure Trace

485M of 823M



Agent-oriented Software Engineering approach

Agent Models and Programming



Requirements

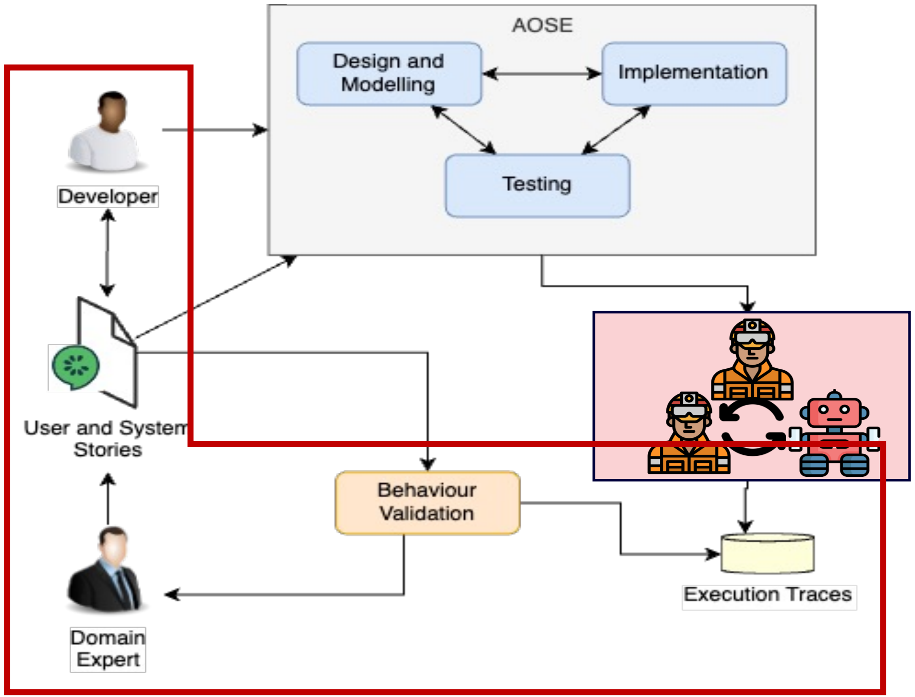
Runtime

Testing, Evaluation, Verification and Validation

Agent-oriented Software Engineering approach

Agent Models and Programming

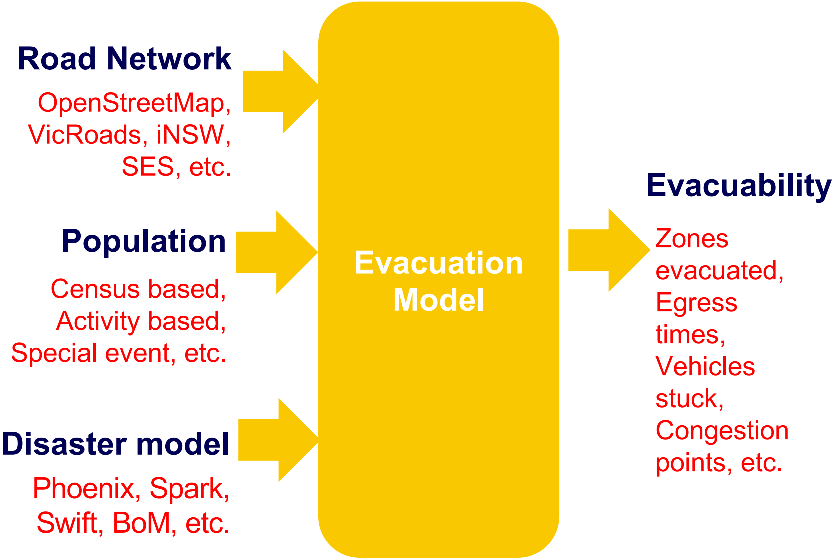
Requirements



Runtime

Testing, Evaluation, Verification and Validation

Case Study: Evacuation Modelling



Road Network

OpenStreetMap, VicRoads, iNSW, SES, etc.

Population

Census based, Activity based, Special event, etc.

Disaster model

Phoenix, Spark, Swift, BoM, etc.

Evacuation Model

Evacuability

Zones evacuated, Egress times, Vehicles stuck, Congestion points, etc.

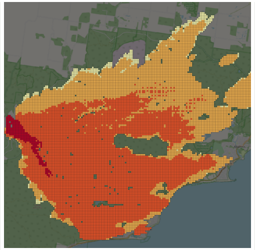
- ❖ Applied work in evacuation modelling for natural disasters, esp. bushfires and floods spanning 10 years
- ❖ Combines agent-based modelling and simulation with **belief-desire-intention** for cognitive reasoning
- ❖ Key stakeholders include Emergency Management VIC, Department of Premier and Cabinet VIC, Department of Land, Water, and Planning, and various councils
- ❖ Funded by CSIRO's Dta61 (2018 - ongoing) [Singh et al.]

2015 // Warrandyte VIC

Is the bridge a choke point for large evacuations?

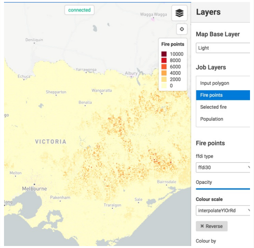
2020 // Sydney NSW

Nepean-Hawkesbury flood evacuation modelling



2021 // VIC

Web-based Evacuation Decision Support Tool for Shires



Map Base Layer: Light

Fire points

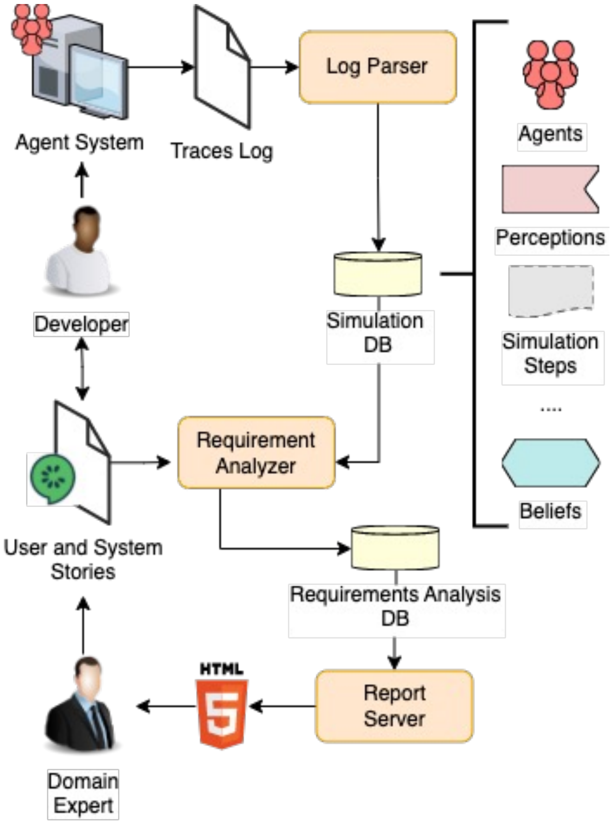
ffdi type: ffdi30

Colour scale: interpolateYlOrRd

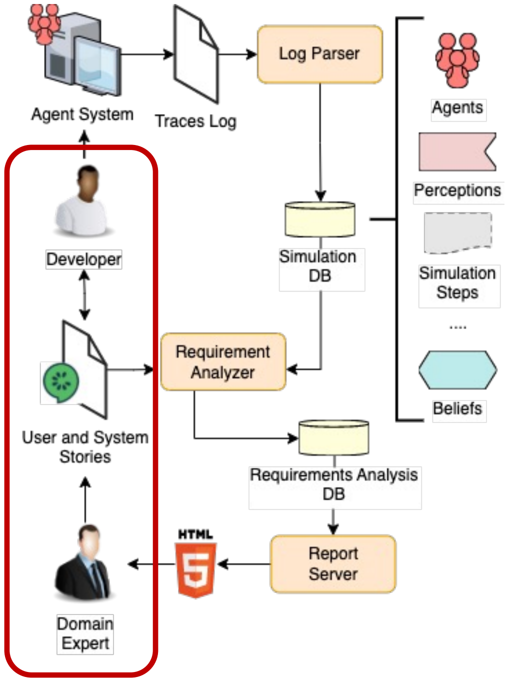
2021 // VIC & WA

State-wide evacuation risk hotspot identification tool

Process Overview



USS and Acceptance Criteria



Feature:
Handling of dependents for full-time residents

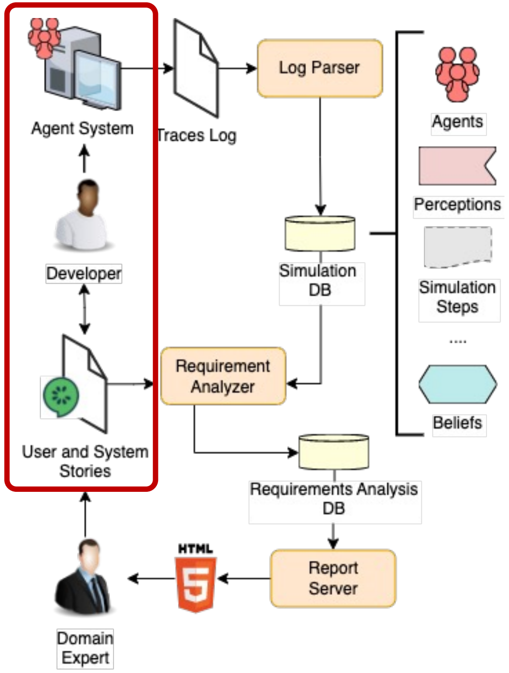
As ResidentFullTime,
I want to always attend to my dependents
so that they are safe

Scenario: first response is always to attend to dependents
Given agent is type ResidentFullTime
Given it believes HasDependents is true
When it believes current_goal is GoalInitialResponse
Then eventually it believes status is to:DependentsPlace

Scenario: ….

Scenario: ….

USS and Acceptance Criteria



As ResidentFullTime,

I want to attend to my dependents,

So that they are safe.

Who

Resident FullTime

<agent/role/system module>

What

Attend Dependents

<achieve or maintain goal>
<do action>

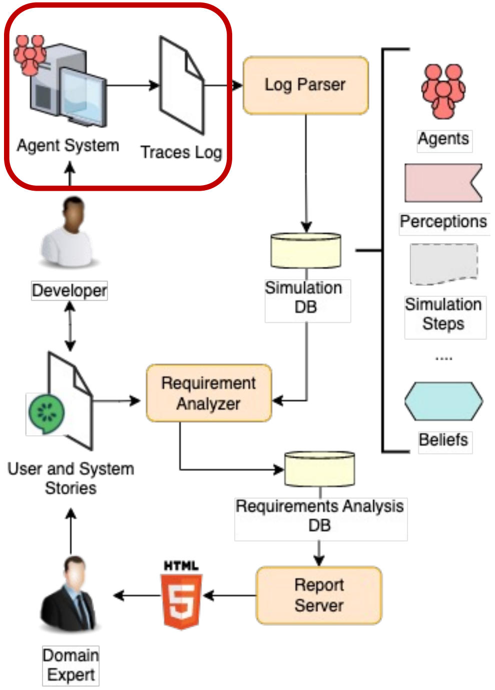
Why

Ensure Dependents Safety

<achieve or maintain goal>
<handle perception>

User and System Stories (AAMAS’21)

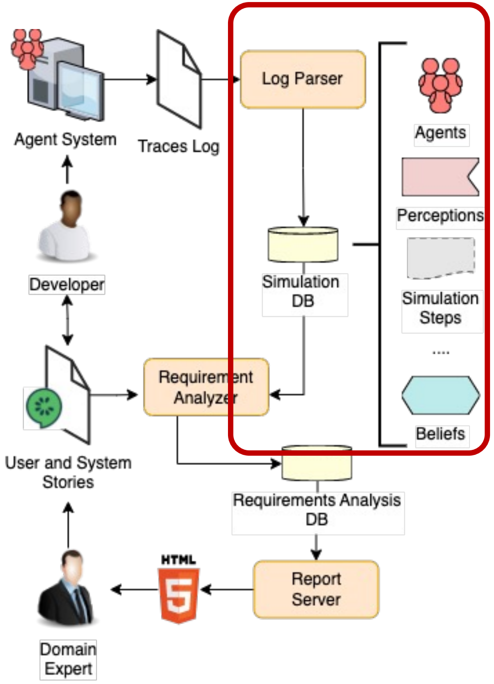
Process Overview

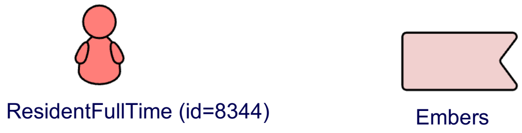


```
42930|11:55:30|ResidentFullTime|8344|saw embers
42930|11:55:30|ResidentFullTime|8344|believes anxietyFromSituation=0.3
42930|11:55:30|ResidentFullTime|8344|believes …
```

```
...|thinks GoalFullResponse~>PlanFullResponse is applicable
...|thinks GoalInitialResponse~>PlanResponseWhenDependentsAfar is not applicable
...|thinks GoalInitialResponse~>PlanResponseWhenDependentsNearby is applicable
...|thinks GoalInitialResponse~>PlanResponseWithoutDependents is not applicable
...|thinks GoalInitialResponse~>PlanDoNothing is applicable
...
```

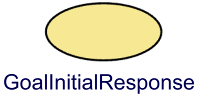
Process Overview

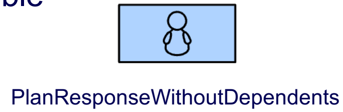




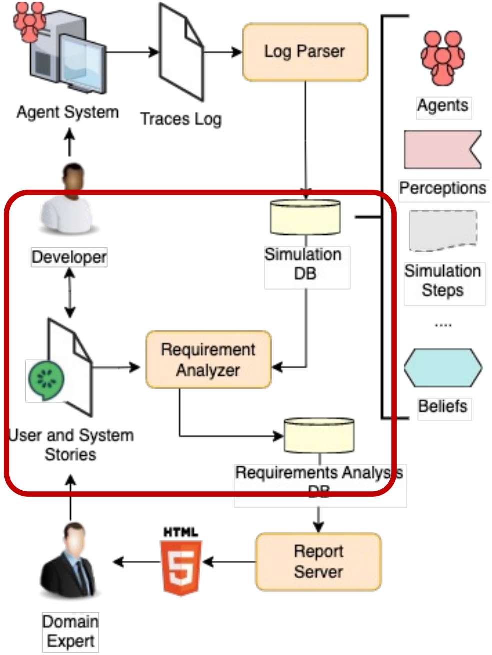
```
42930|11:55:30|ResidentFullTime|8344|saw embers
42930|11:55:30|ResidentFullTime|8344|believes anxietyFromSituation=0.3
42930|11:55:30|ResidentFullTime|8344|believes …
```

```
...|thinks GoalFullResponse~>PlanFullResponse is applicable
...|thinks GoalInitialResponse~>PlanResponseWhenDependentsAfar is not applicable
...|thinks GoalInitialResponse~>PlanResponseWhenDependentsNearby is applicable
...|thinks GoalInitialResponse~>PlanResponseWithoutDependents is not applicable
...|thinks GoalInitialResponse~>PlanDoNothing is applicable
...
```



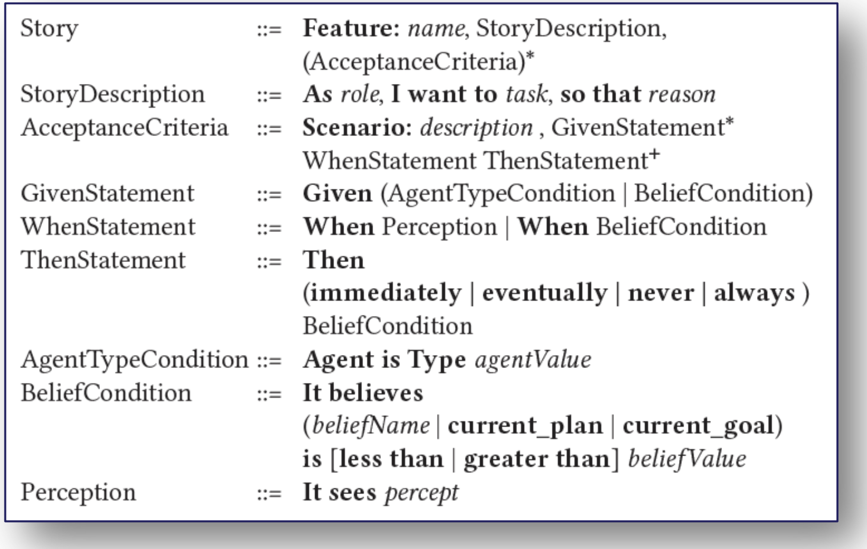


Requirement Analysis



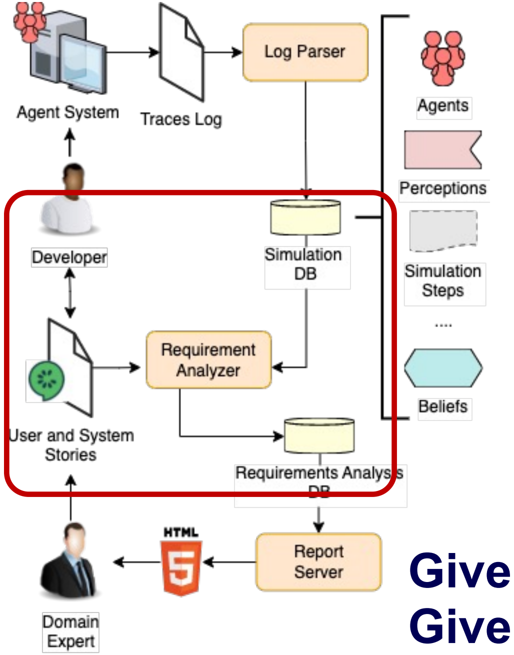
Extension Gherkin Syntax
Agent-specific and temporal constructs
Implemented on top of proven BDD testing framework

System Story Extension BNF

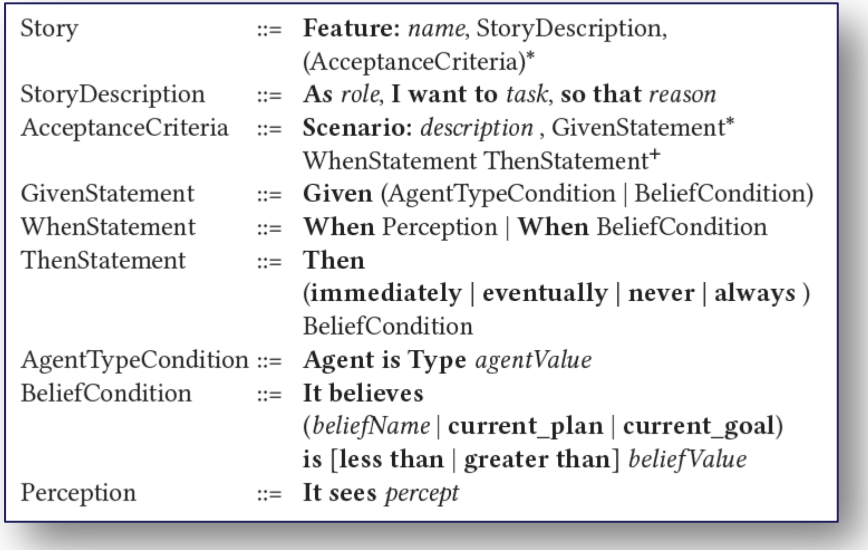


Story	::=	**Feature:** *name*, StoryDescription, (AcceptanceCriteria)*
StoryDescription	::=	**As** *role*, **I want to** *task*, **so that** *reason*
AcceptanceCriteria	::=	**Scenario:** *description* , GivenStatement* WhenStatement ThenStatement$^+$
GivenStatement	::=	**Given** (AgentTypeCondition | BeliefCondition)
WhenStatement	::=	**When** Perception | **When** BeliefCondition
ThenStatement	::=	**Then** (**immediately** | **eventually** | **never** | **always**) BeliefCondition
AgentTypeCondition	::=	**Agent is Type** *agentValue*
BeliefCondition	::=	**It believes** (*beliefName* | **current_plan** | **current_goal**) **is** [**less than** | **greater than**] *beliefValue*
Perception	::=	**It sees** *percept*

Requirement Analysis



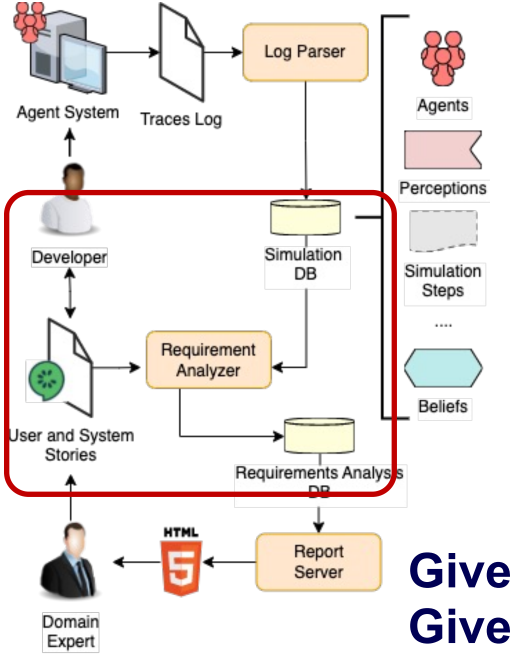
System Story Extension BNF



Story	::=	**Feature:** *name*, StoryDescription, (AcceptanceCriteria)*
StoryDescription	::=	**As** *role*, **I want to** *task*, **so that** *reason*
AcceptanceCriteria	::=	**Scenario:** *description* , GivenStatement* WhenStatement ThenStatement⁺
GivenStatement	::=	**Given** (AgentTypeCondition | BeliefCondition)
WhenStatement	::=	**When** Perception | **When** BeliefCondition
ThenStatement	::=	**Then** (**immediately** | **eventually** | **never** | **always**) BeliefCondition
AgentTypeCondition	::=	**Agent is Type** *agentValue*
BeliefCondition	::=	**It believes** (*beliefName* | **current_plan** | **current_goal**) **is** [**less than** | **greater than**] *beliefValue*
Perception	::=	**It sees** *percept*

Given agent is type ResidentFullTime
Given it believes HasDependents is true
When it believes current_goal is GoalInitialResponse
Then eventually it believes status is to:DependentsPlace

Requirement Analysis

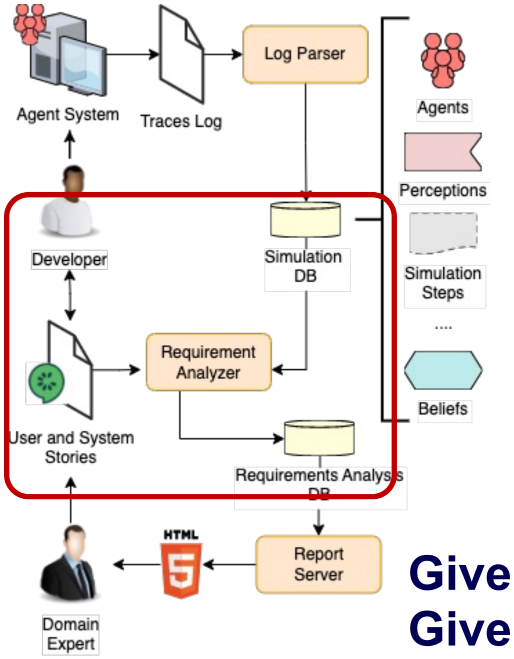


Fault Model

Fault Name	Fault Type	Interpretation
PASS		Trigger observed, conditions met, and the observed behaviour of the agent complies with the specification
FAIL	Strong	Trigger observed, conditions met, but the observed behaviour of the agent does not comply with the specification.
NO_TRIGGER	Weak	Trigger (perception or belief update) was not observed for any agent in the simulation
TRIGGERED BUT_GIVEN NOT_MET	Weak	Trigger observed, but belief state of the agent did not meet the given conditions

Given agent is type ResidentFullTime
Given it believes HasDependents is true
When it believes current_goal is GoalInitialResponse
Then eventually it believes status is to:DependentsPlace

Requirement Analysis



Fault Model

Fault Name	Fault Type	Interpretation
PASS		Trigger observed, conditions met, and the observed behaviour of the agent complies with the specification
FAIL	Strong	Trigger observed, conditions met, but the observed behaviour of the agent does not comply with the specification.
NO_TRIGGER	Weak	Trigger (perception or belief update) was not observed for any agent in the simulation
TRIGGERED BUT_GIVEN NOT_MET	Weak	Trigger observed, but belief state of the agent did not meet the given conditions

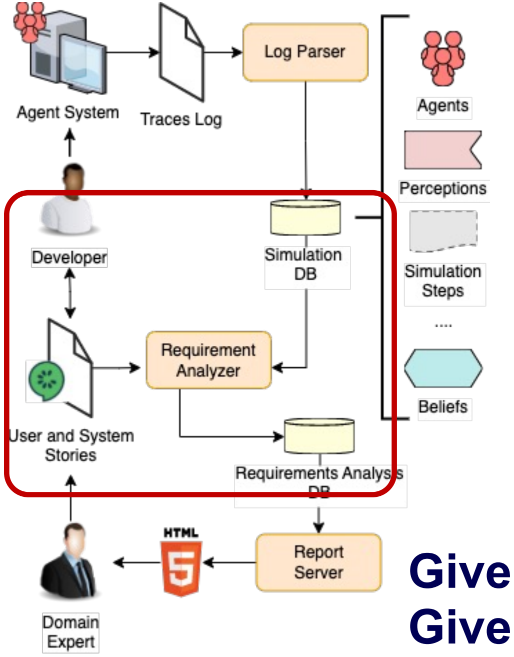
Given agent is type ResidentFullTime ✓

Given it believes HasDependents is true ✓

When it believes current_goal is GoalInitialResponse ✓

Then eventually it believes status is to:DependentsPlace ✓

Requirement Analysis

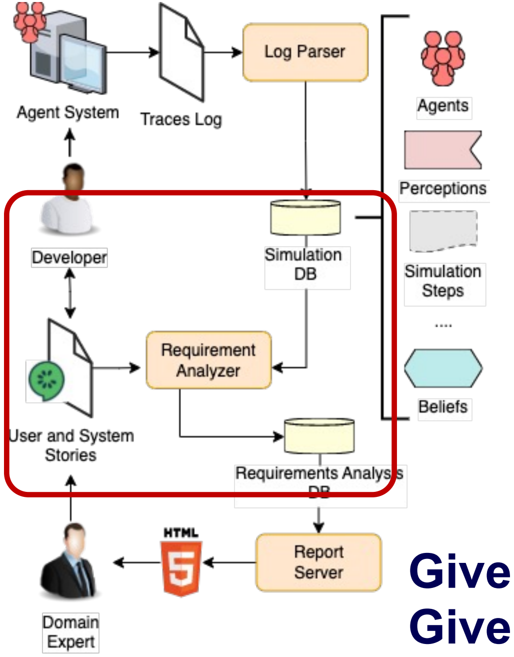


Fault Model

Fault Name	Fault Type	Interpretation
PASS		Trigger observed, conditions met, and the observed behaviour of the agent complies with the specification
FAIL	Strong	Trigger observed, conditions met, but the observed behaviour of the agent does not comply with the specification.
NO_TRIGGER	Weak	Trigger (perception or belief update) was not observed for any agent in the simulation
TRIGGERED BUT_GIVEN NOT_MET	Weak	Trigger observed, but belief state of the agent did not meet the given conditions

Given agent is type ResidentFullTime ✓
Given it believes HasDependents is true ✓
When it believes current_goal is GoalInitialResponse ✓
Then eventually it believes status is to:DependentsPlace ✗

Requirement Analysis

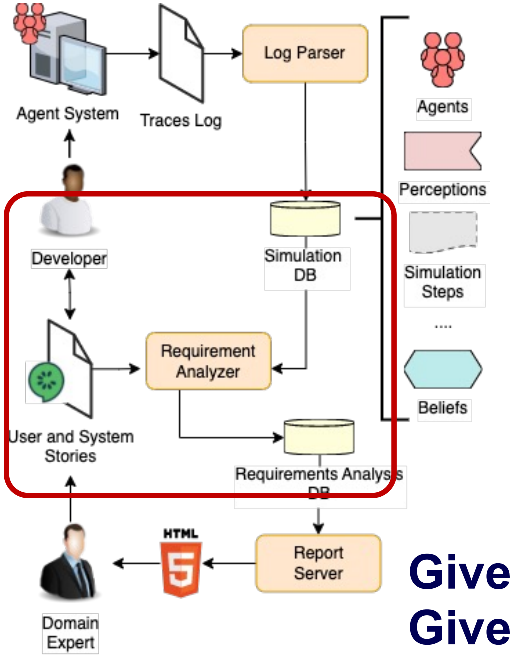


Fault Model

Fault Name	Fault Type	Interpretation
PASS		Trigger observed, conditions met, and the observed behaviour of the agent complies with the specification
FAIL	Strong	Trigger observed, conditions met, but the observed behaviour of the agent does not comply with the specification.
NO_TRIGGER	Weak	Trigger (perception or belief update) was not observed for any agent in the simulation
TRIGGERED BUT_GIVEN NOT_MET	Weak	Trigger observed, but belief state of the agent did not meet the given conditions

Given agent is type ResidentFullTime ✓
Given it believes HasDependents is true ✓
When it believes current_goal is GoalInitialResponse ✗
Then eventually it believes status is to:DependentsPlace ?

Requirement Analysis

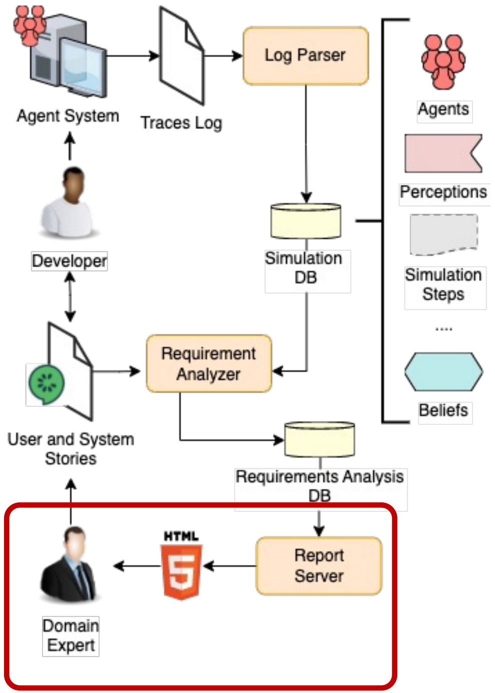


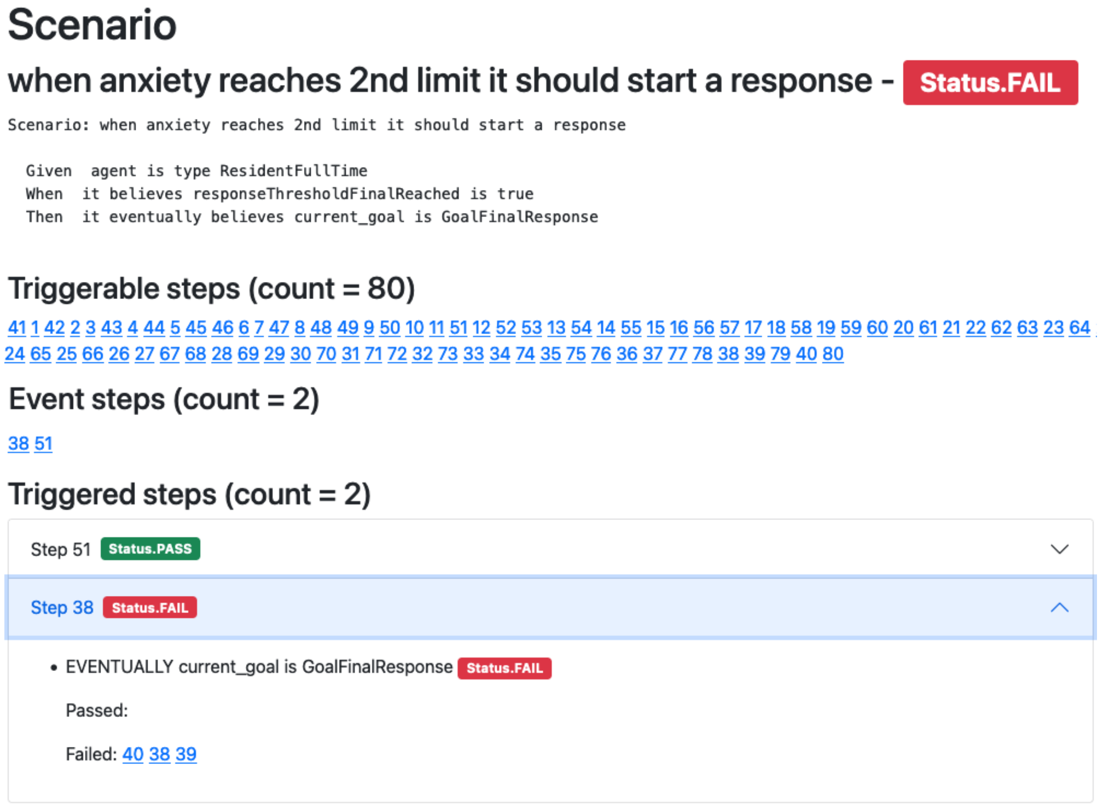
Fault Model

Fault Name	Fault Type	Interpretation
PASS		Trigger observed, conditions met, and the observed behaviour of the agent complies with the specification
FAIL	Strong	Trigger observed, conditions met, but the observed behaviour of the agent does not comply with the specification.
NO_TRIGGER	Weak	Trigger (perception or belief update) was not observed for any agent in the simulation
TRIGGERED BUT_GIVEN NOT_MET	Weak	Trigger observed, but belief state of the agent did not meet the given conditions

Given agent is type ResidentFullTime ✓
Given it believes HasDependents is true ✗
When it believes current_goal is GoalInitialResponse ✓
Then eventually it believes status is to:DependentsPlace ?

Process Overview





Scenario

when anxiety reaches 2nd limit it should start a response - Status.FAIL

```
Scenario: when anxiety reaches 2nd limit it should start a response

  Given  agent is type ResidentFullTime
  When  it believes responseThresholdFinalReached is true
  Then  it eventually believes current_goal is GoalFinalResponse
```

Triggerable steps (count = 80)

41 1 42 2 3 43 4 44 5 45 46 6 7 47 8 48 49 9 50 10 11 51 12 52 53 13 54 14 55 15 16 56 57 17 18 58 19 59 60 20 61 21 22 62 63 23 64
24 65 25 66 26 27 67 68 28 69 29 30 70 31 71 72 32 73 33 34 74 35 75 76 36 37 77 78 38 39 79 40 80

Event steps (count = 2)

38 51

Triggered steps (count = 2)

Step 51 Status.PASS

Step 38 Status.FAIL

- EVENTUALLY current_goal is GoalFinalResponse Status.FAIL

  Passed:

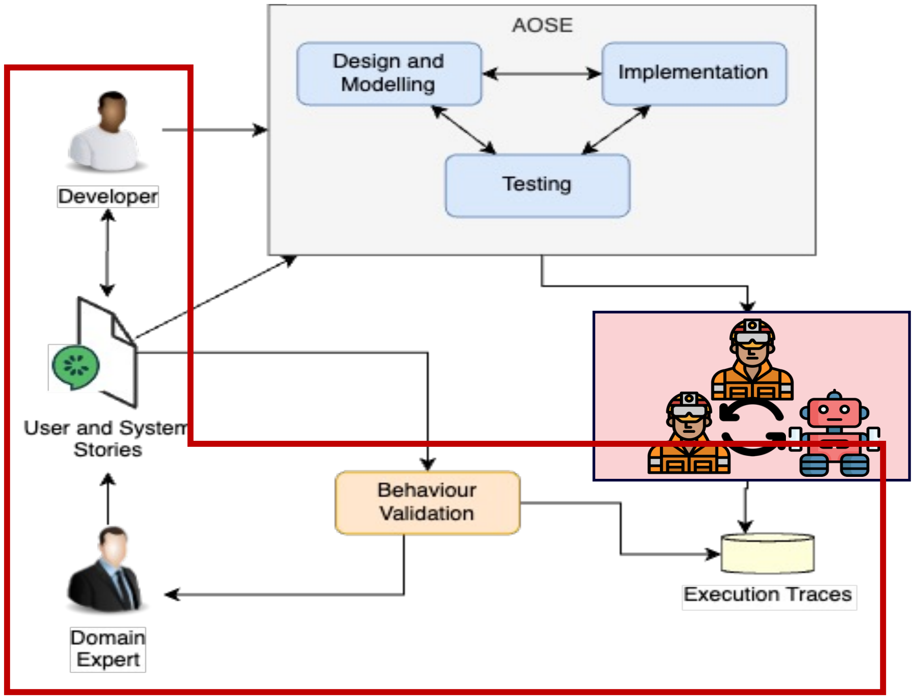
  Failed: 40 38 39

Agent-oriented Software Engineering approach

Agent Models and Programming

Requirements

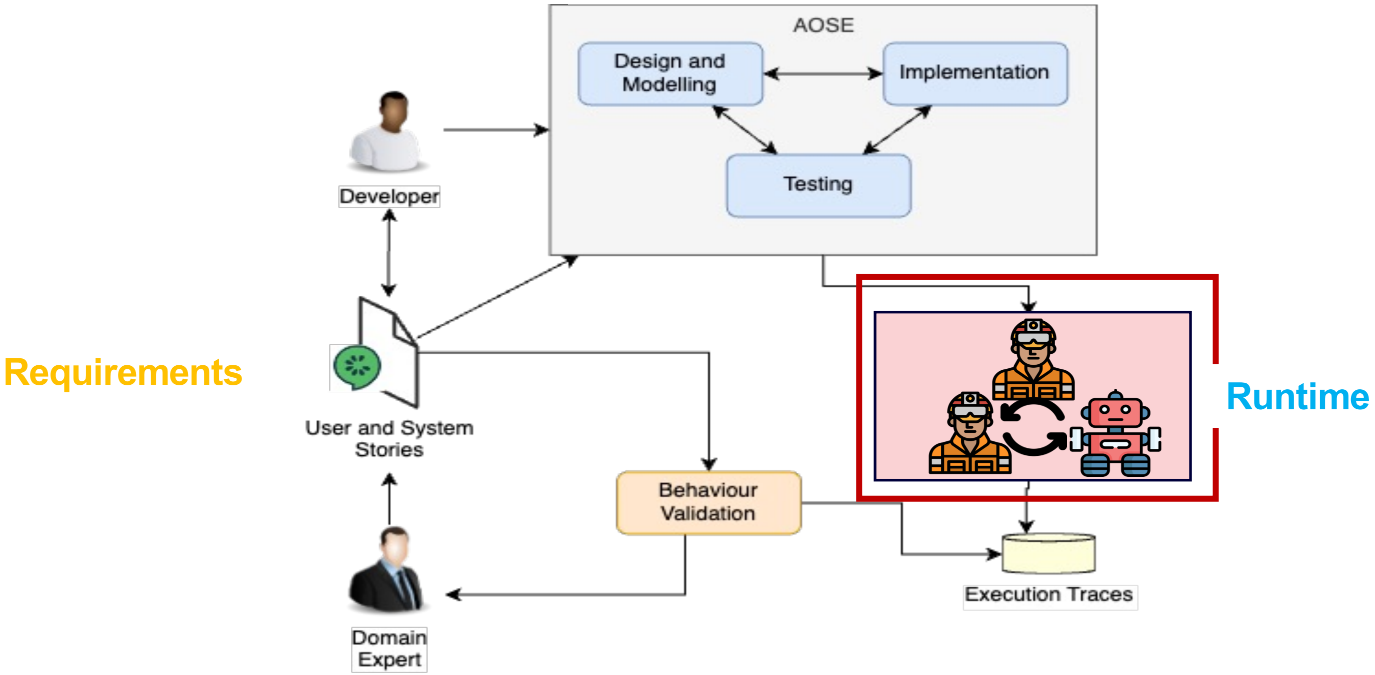
Runtime



Testing, Evaluation, Verification and Validation

Agent-oriented Software Engineering approach

Agent Models and Programming



Testing, Evaluation, Verification and Validation

SARL Goal Engine

Goal oriented reasoning

- Goals: Achievement; Maintenance, ...
- Plans: Actions failures and durations
- Beliefs

Reasoning

- Customizable Goal / Plan Selection
- Customizable Intention Scheduling
- Goal achievement verification
- Meta reasoning (e.g. valuings)

```
skill ExploreArea extends Goal implements AchievementGoal{
  uses SearchRescueBeliefs, DroneState
  def context : boolean {…}
  def success : boolean {
    explorationRate(currentArea) >= 0.95f
  }
  def failure : boolean {…}
}
```

SARL Programming Language

http://www.sarl.io

- Agent architecture-agnostic
- Powerful (yet simple) extension mechanism
- Distribution (network) abstraction

Open-Source Project
Full IDE Support
Compatible with modern deployment tools
Java interoperability

Search and Rescue Scenario

Drones assist in locating and identifying victims, via tasks assigned to them by the human drone operator which they carry out autonomously.

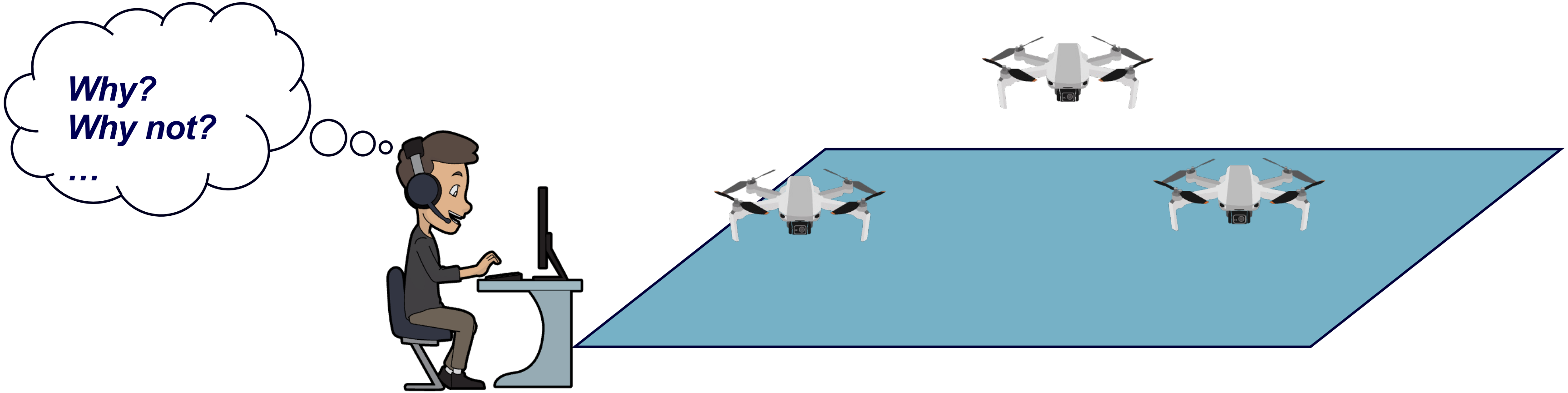
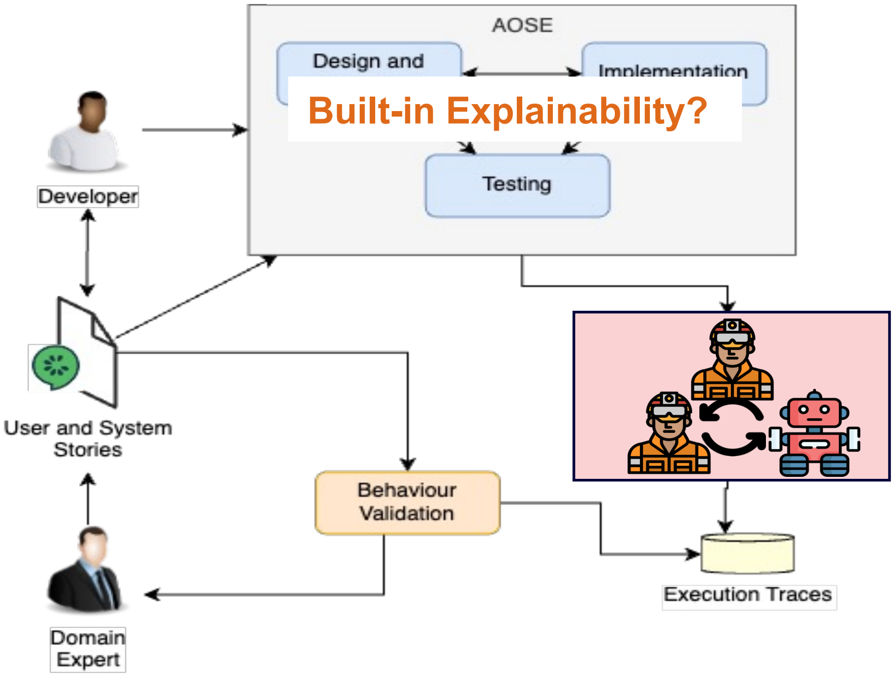


Image: wikimedia

Agent-oriented Software Engineering approach

Agent Models and Programming

Requirements



Built-in Explainability?

Runtime

Testing, Evaluation, Verification and Validation

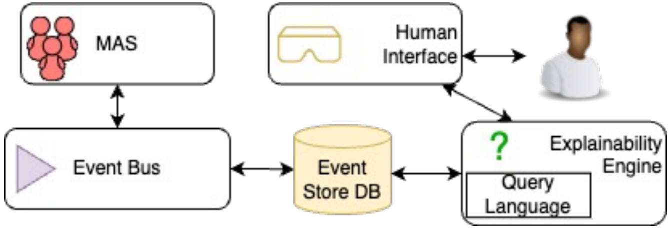
Explainable Agents (XAg) by design

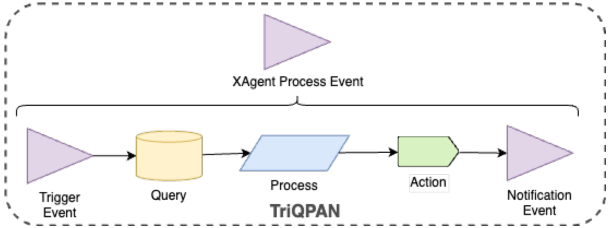
AAMAS’24 (Main Track; Blue Sky)

Explainability is an essential feature for Trust

eXplainable-by-design Agents (XAg)

- Event driven architecture
- Explainable decision-making processes
  - TriQPAN Design Pattern (AAMAS’24 Main Track) - Wednesday
- Query languages and explanation engines





Research agenda: Challenges and opportunities

- AAMAS’24 Blue Sky - Friday

Agile AOSE

Requirements that are understandable and traceable

- Use main steam SE practices
- Link requirements to system component (no black box)

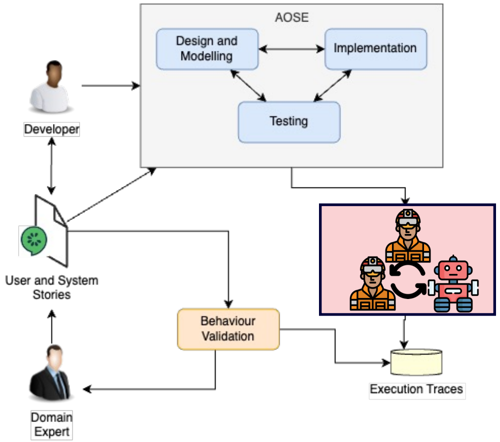
Testable and Verifiable Intelligent Systems

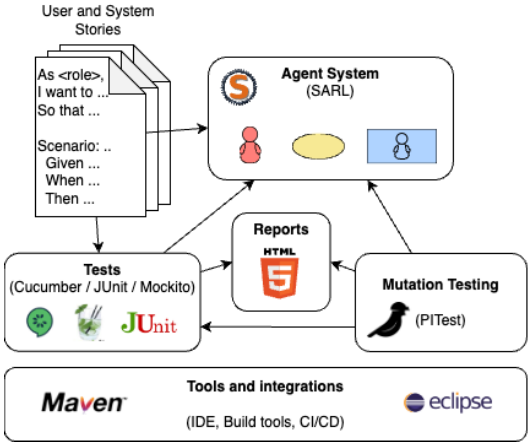
- Validate System behaviors against requirements
- Testing frameworks for independent modules
- Validation of testing quality

Programable using concepts familiar to humans

- Goal oriented practical reasoning

Explainable-by-design agents (XAg)





AAMAS, 2021; AAMAS, 2022; AAMAS, 2023; AAMAS, 2024

Looking Forward …

Agile practices for AOSE

- Every step for the SDLC (ES; DDD; CI/CD; etc.)
- Agile methodologies

Design and Architectures

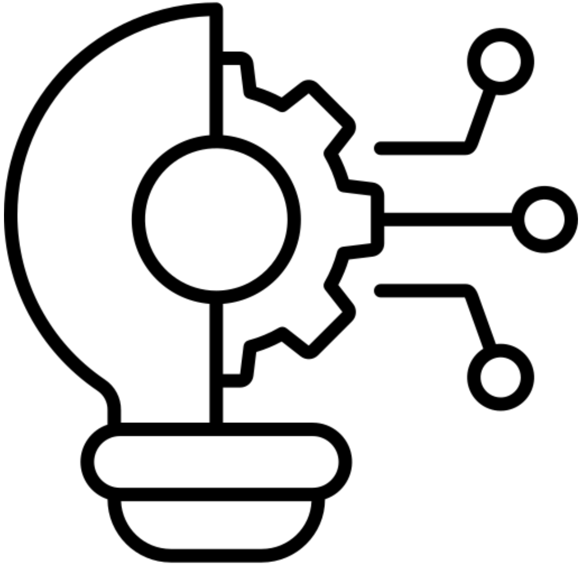
- DDD; MDE; Event-Driven architectures
- Design Patterns and Explainable-by-design

Test, Evaluation, Verification and Validation

Agent for mainstream SE projects

Models; Programming; …

Tools and infrastructure support



THANK YOU!

Agile Approach for Agent Oriented Software Engineering

Sebastian Rodriguez

sebastian.rodriguez@rmit.edu.au

RMIT University – CIAIRI

EMAS Workshop @ AAMAS'24

7 May 2024

Auckland, NZ

Papers

(AAMAS21) Rodriguez, S., Thangarajah, J. and Winikoff, M. (2021) ‘User and System Stories: An Agile Approach for Managing Requirements in AOSE’, in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems (AAMAS ’21), pp. 1064–1072. Available at: https://doi.org/10.5555/3461017.3461136.

(AAMAS22) Rodriguez, S. *et al.* (2022) ‘Testing Requirements via User and System Stories in Agent Systems’, in *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems (AAMAS ’22), pp. 1119–1127. Available at: https://ifaamas.org/Proceedings/aamas2022/pdfs/p1119.pdf (Accessed: 12 May 2022).

(AAMAS23) Rodriguez, S., Thangarajah, J. and Winikoff, M. (2023) ‘A Behaviour-Driven Approach for Testing Requirements via User and System Stories in Agent Systems’, in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems (AAMAS ’23), pp. 1182–1190. Available at: https://doi.org/10.5555/3545946.3598761.

(AAMAS24 a) Rodriguez, S., Thangarajah, J. and Davey, A. (2024) ‘Design Patterns for Explainable Agents (XAg)’, in *Proceedings of the 2024 International Conference on Autonomous Agents and Multiagent Systems*. Auckland, New Zeland (AAMAS ’24).

(AAMAS24 b) Rodriguez, S. and Thangarajah, J. (2024) ‘Explainable Agents (XAg) by Design’, in *Proceedings of the 2024 International Conference on Autonomous Agents and Multiagent Systems (Blue Sky)*. Auckland, New Zeland (AAMAS ’24)