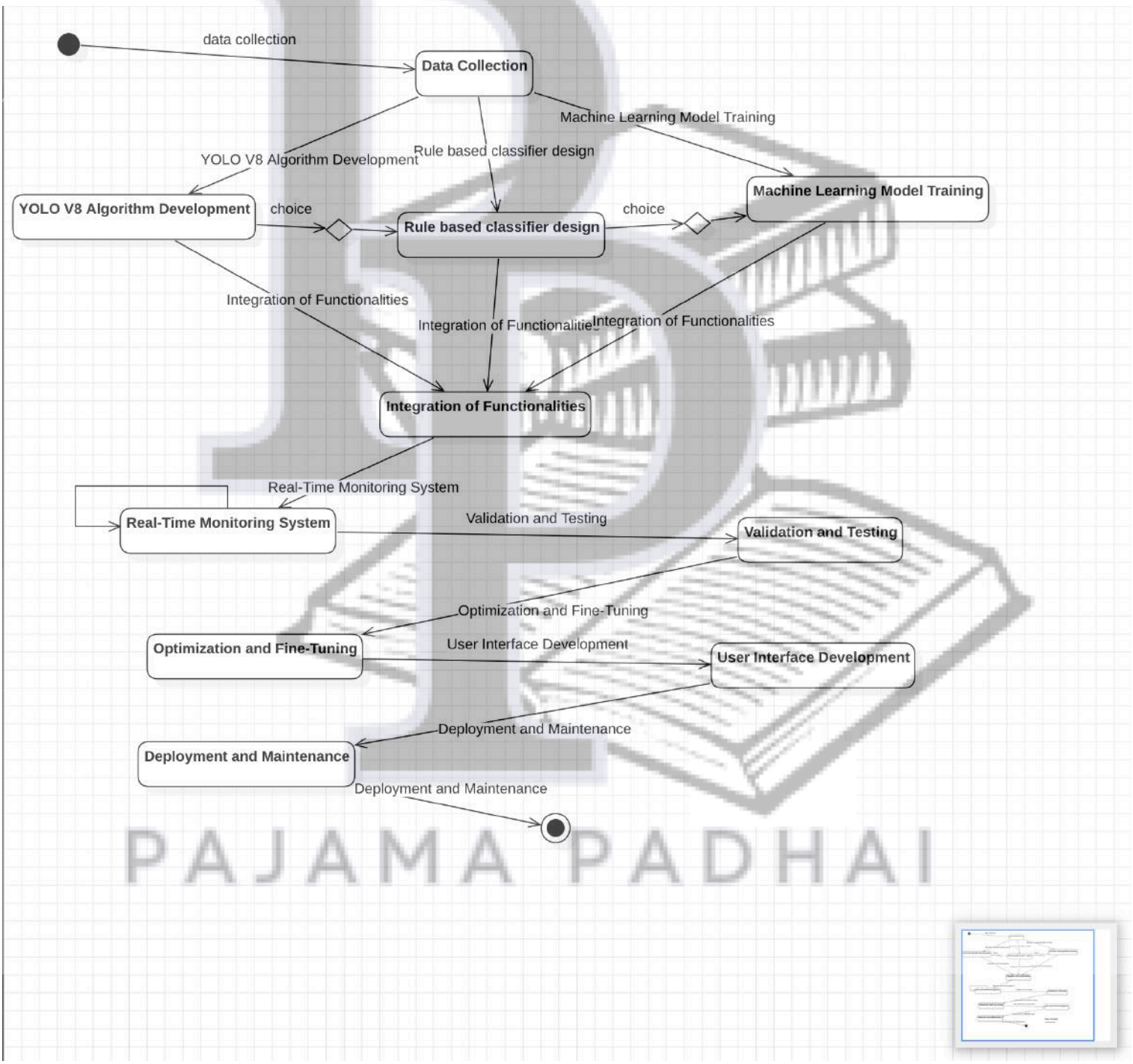
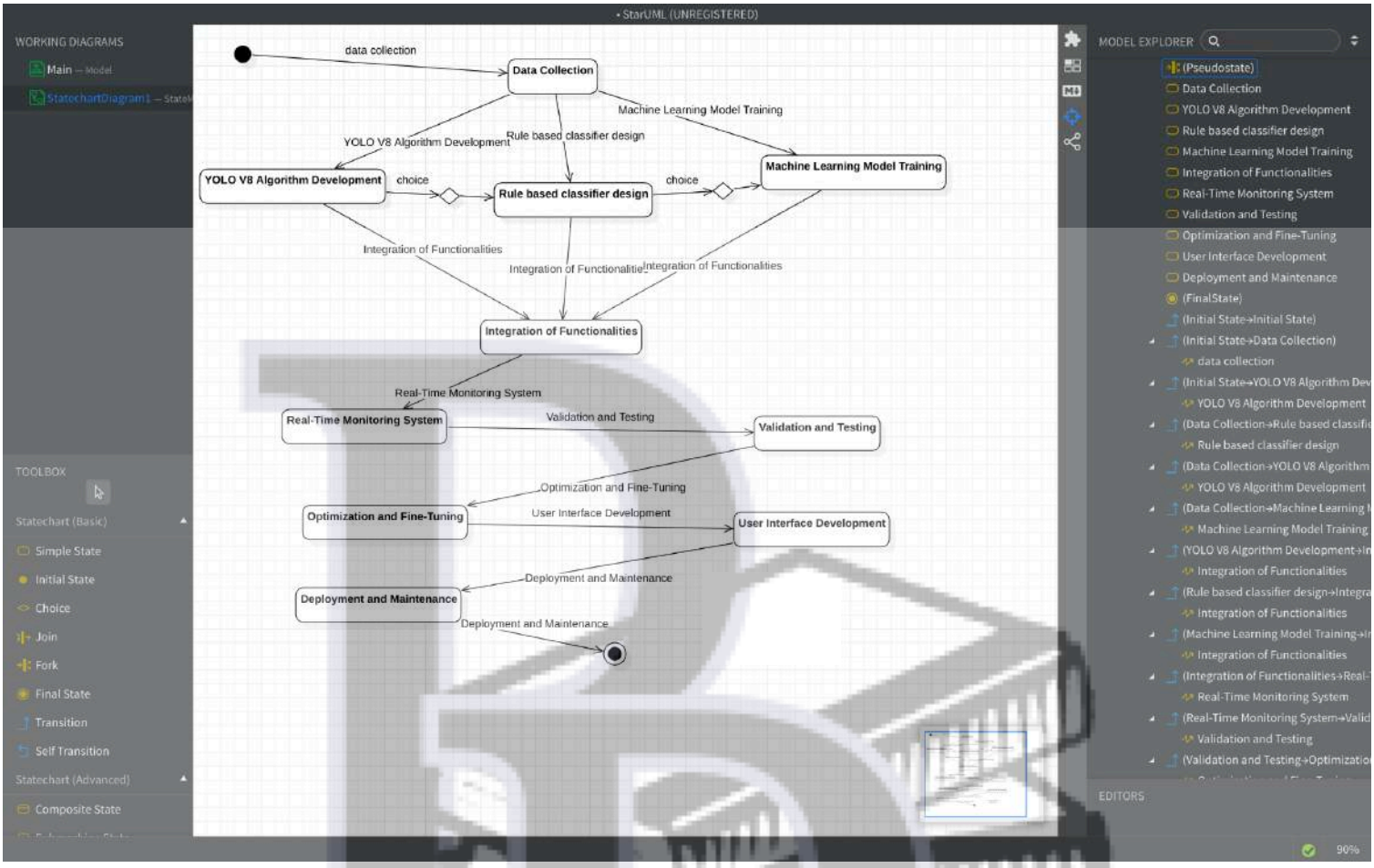
Software Engineering Lab
Assignment 3

1. State Transition Diagram





To create a State Transition Diagram for the given project, we need to identify the different states and transitions between them. Here's a breakdown based on the provided information:

States:

- Data Collection
- YoloV8 Algorithm Development
- Rule-Based Classifier Design
- Machine Learning Model Training
- Integration of Functionalities
- Real-Time Monitoring System
- Validation and Testing
- Optimization and Fine-Tuning
- User Interface Development
- Deployment and Maintenance

Transitions:

- Data Collection -> YoloV8 Algorithm Development
- Data Collection -> Rule-Based Classifier Design
- Data Collection -> Machine Learning Model Training
- YoloV8 Algorithm Development -> Integration of Functionalities
- Rule-Based Classifier Design -> Integration of Functionalities
- Machine Learning Model Training -> Integration of Functionalities
- Integration of Functionalities -> Real-Time Monitoring System
- Real-Time Monitoring System -> Validation and Testing
- Validation and Testing -> Optimization and Fine-Tuning
- Optimization and Fine-Tuning -> User Interface Development
- User Interface Development -> Deployment and Maintenance

Additional Elements:

- Initial State: Data Collection
- Final State: Deployment and Maintenance
- Choice: The choice between YoloV8 Algorithm Development, Rule-Based Classifier Design, and Machine Learning Model Training after Data Collection.
- Join: Integration of Functionalities (where all three functionalities merge into one)
- Fork: After Optimization and Fine-Tuning, branching into User Interface Development and Deployment and Maintenance.

Self-Transition:
Real-Time Monitoring System -> Real-Time Monitoring System (for continuous analysis)

CREATING STATE TRANSITION DIAGRAM ON STARUML:

To create a Statechart Diagram:

1. Select first an element where a new Statechart Diagram to be contained as a child.
2. Select **Model | Add Diagram | Statechart Diagram** in Menu Bar or select **Add Diagram | Statechart Diagram** in Context Menu.

To create a Simple State:

1. Select **Simple State** in **Toolbox**.
2. Drag on the diagram as the size of Simple State.

To create a Composite State:

1. Select **Composite State** in **Toolbox**.
2. Drag on the diagram as the size of Composite State.

To create a Submachine State:

1. Select **Submachine State** in **Toolbox**.
2. Drag on the diagram as the size of Submachine State.
3. Select a StateMachine in **Element Picker Dialog**.

To create an Orthogonal State:

1. Select **Orthogonal State** in **Toolbox**.
2. Drag on the diagram as the size of Orthogonal State.

You can use **QuickEdit** for State by double-click or press `Enter` on a selected State.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

```
expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
stereotype ::= (identifier)
visibility ::= '+' | '#' | '-' | '~'
name ::= (identifier)
```

- **Add ConnectionPointReference** : Add a connection point reference.
- **Add Region** : Add a region.
- **Add Note** : Add a linked note.
- **Add Constraint** : Add a constraint.
- **Add Entry Activity** : Add an entry activity.
- **Add Do Activity** : Add an do activity.
- **Add Exit Activity** : Add an exit activity.
- **Add Internal Transition** : Add an internal transition.

To add an Entry Activity:

1. Select a State.
2. Select **Model | Add | Entry Activity** in Menu Bar or **Add | Entry Activity** in Context Menu.
3. Select a kind of Activity to create (one of OpaqueBehavior, Activity, StateMachine, or Interaction).

To add a Do Activity:

1. Select a State.
2. Select **Model | Add | Do Activity** in Menu Bar or **Add | Do Activity** in Context Menu.
3. Select a kind of Activity to create (one of OpaqueBehavior, Activity, StateMachine, or Interaction).

To add an Exit Activity:

1. Select a State.
2. Select **Model | Add | Exit Activity** in Menu Bar or **Add | Exit Activity** in Context Menu.
3. Select a kind of Activity to create (one of OpaqueBehavior, Activity, StateMachine, or Interaction).

To add an Internal Transition:

1. Select a State.
2. Popup Quic Edit for State by double click or press `Enter` on a selected State.
3. Select **Add Internal Transition** button in Quick Edit.

You can use **QuickEdit** for Internal Transition by double-click or press `Enter` on a selected Internal Transition.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

  ```
  expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  ```

- **Add Trigger Event** : Add a trigger event.
- **Add Effect Behavior** : Add an effect behavior.

To add a Region:

1. Select a State.
2. Select **Model | Add | Region** in Menu Bar or **Add | Region** in Context Menu.

To create a Initial State:

1. Select **Initial State** in **Toolbox.**
2. Click at the position on the diagram.

To create a Choice:

1. Select **Choice** in **Toolbox**.
2. Click at the position on the diagram.

To create a Fork:

1. Select **Fork** in **Toolbox**.
2. Drag on the diagram as the size of Fork.

To create a Transition (or Self Transition):

1. Select **Transition** (or **Self Transition**) in **Toolbox**.
2. Drag from a State and drop on another State. (Just click on a State if you want to create a Self Transition.)

You can use **QuickEdit** for Transition by double-click or press `Enter` on a selected Transition.

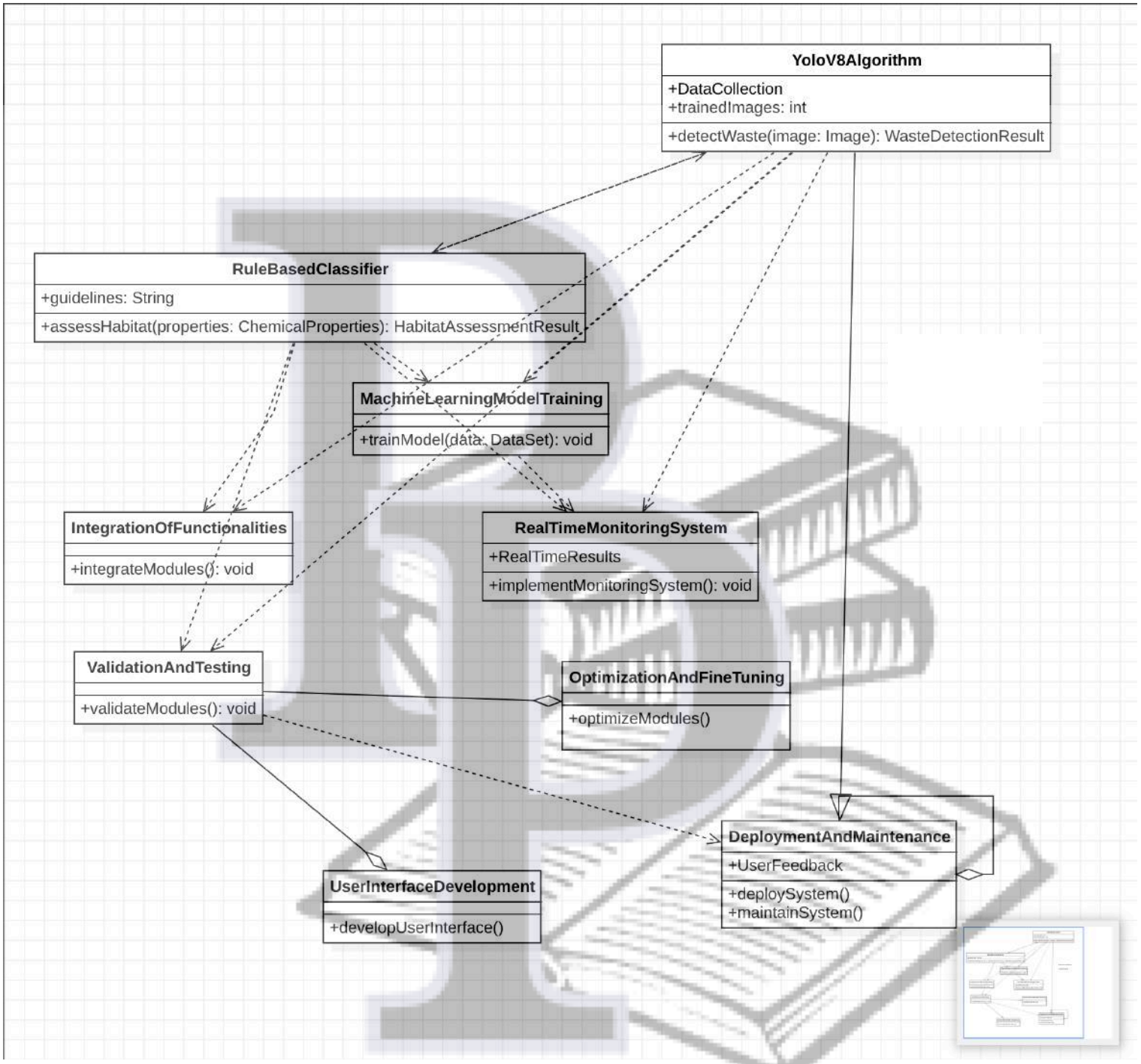
- **Transition Expression** : Edit transition expression.

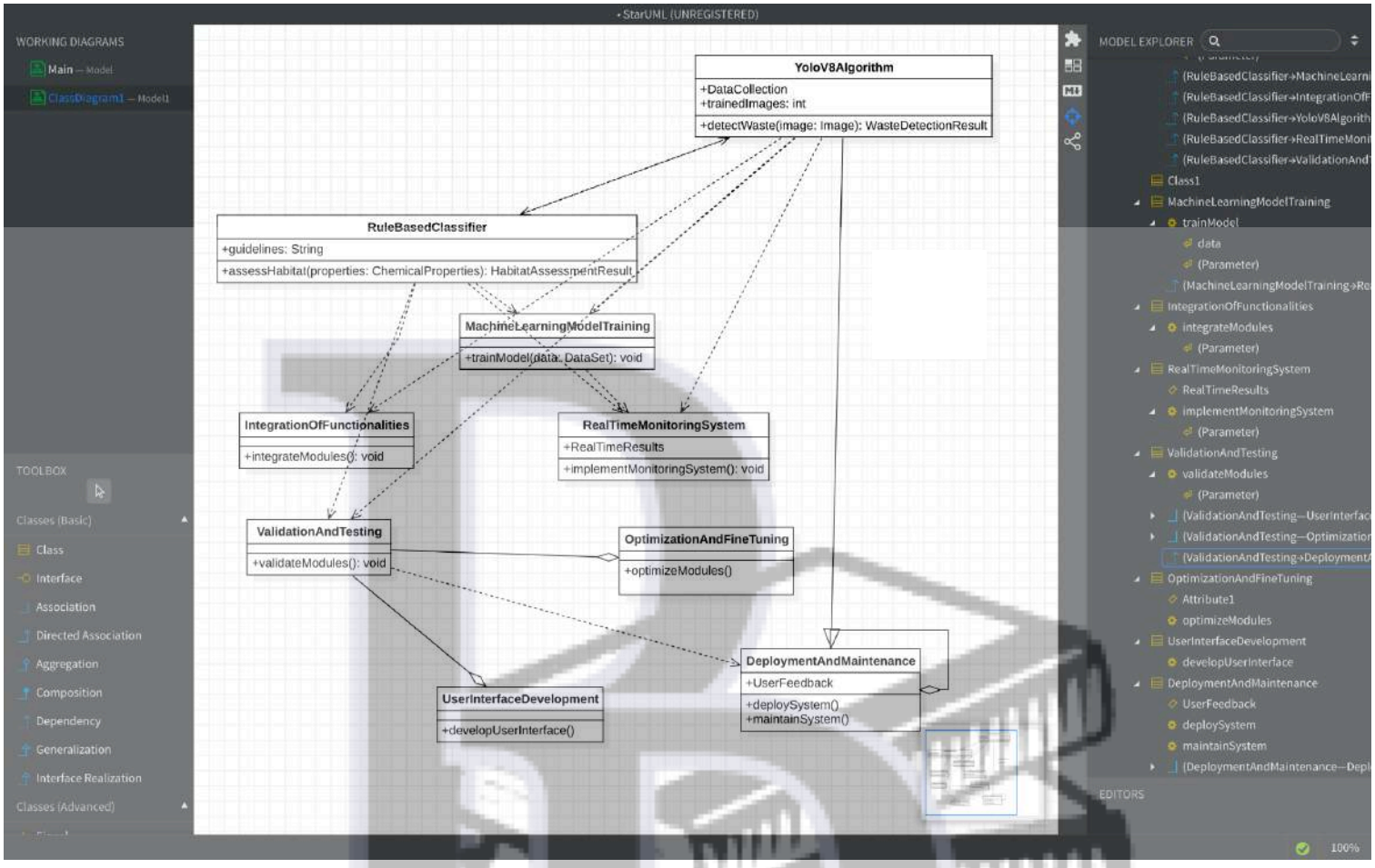
  Syntax of Transition Expression

  ```
  transition ::= [ trigger-list ] [ '[' guard ']' ] [ '/' effect ]
  trigger-list ::= trigger [ ',' trigger ]
  trigger ::= (identifier)
  guard ::= (string)
  effect ::= (identifier)
  ```

- **Add Note** : Add a linked note.
- **Add Constraint** : Add a constraint.
- **Add Trigger Event** : Add a trigger event.
- **Add Effect Behavior** : Add an effect behavior.

2. Class Diagram





Classes:

YoloV8Algorithm
Attributes:
+ trainedImages: int
Methods:
+ detectWaste(image: Image): WasteDetectionResult

RuleBasedClassifier
Attributes:
+ guidelines: String
Methods:
+ assessHabitat(properties: ChemicalProperties): HabitatAssessmentResult

MachineLearningModel
Attributes:
+ trainingDataSize: int
Methods:
+ classifyWaterQuality(data: DataSet): WaterQualityClassification

DataCollection
Methods:
+ gatherUnderwaterImages(): Image[]
+ collectChemicalPropertiesData(): ChemicalProperties[]

YoloV8AlgorithmDevelopment

Methods:
+ implementYoloV8Algorithm(images: Image[]): void
+ trainAlgorithm(): void

RuleBasedClassifierDesign
Methods:
+ developClassifier(guidelines: String): void

MachineLearningModelTraining
Methods:
+ trainModel(data: DataSet): void

IntegrationOfFunctionalities
Methods:
+ integrateModules(): void

RealTimeMonitoringSystem
Methods:
+ implementMonitoringSystem(): void

ValidationAndTesting
Methods:
+ validateModules(): void

OptimizationAndFineTuning
Methods:
+ optimizeModules(): void

UserInterfaceDevelopment
Methods:
+ developUserInterface(): void

DeploymentAndMaintenance
Methods:
+ deploySystem(): void
+ maintainSystem(): void

Relationships:

- Dependency: DataCollection depends on YoloV8AlgorithmTrainingData and ChemicalPropertiesData.
- Dependency: YoloV8Algorithm depends on YoloV8AlgorithmTrainingData.
- Dependency: RuleBasedClassifier depends on ChemicalPropertiesData.
- Dependency: MachineLearningModel depends on ChemicalPropertiesData.
- Dependency: IntegrationOfFunctionalities depends on YoloV8Algorithm, RuleBasedClassifier, and MachineLearningModel.
- Dependency: RealTimeMonitoringSystem depends on IntegrationOfFunctionalities.
- Dependency: ValidationAndTesting depends on IntegrationOfFunctionalities.
- Dependency: OptimizationAndFineTuning depends on ValidationAndTesting.
- Dependency: UserInterface depends on RealTimeResults and UserFeedback.
- Dependency: DeploymentAndMaintenance depends on IntegrationOfFunctionalities.

Generalization: None evident in the given project description.

Association:

- YoloV8Algorithm has an association with YoloV8AlgorithmTrainingData.
- RuleBasedClassifier has an association with ChemicalPropertiesData.
- MachineLearningModel has an association with ChemicalPropertiesData.
- RealTimeMonitoringSystem has an association with RealTimeResults and UserFeedback.
- UserInterface has an association with RealTimeResults and UserFeedback.

Multiplicity:
- DataCollection has a 1 to many multiplicity with YoloV8AlgorithmTrainingData and ChemicalPropertiesData.
- YoloV8Algorithm has a 1 to 1 multiplicity with YoloV8AlgorithmTrainingData.
- RuleBasedClassifier has a 1 to 1 multiplicity with ChemicalPropertiesData.
- MachineLearningModel has a 1 to 1 multiplicity with ChemicalPropertiesData.
- RealTimeMonitoringSystem has a 1 to 1 multiplicity with RealTimeResults and UserFeedback.
- UserInterface has a 1 to 1 multiplicity with RealTimeResults and UserFeedback.

Aggregation: None evident in the given project description.

Composition: None evident in the given project description.

CREATING CLASS DIAGRAM ON STARUML:

To create a Class Diagram:

1. First select an element where a new Class Diagram to be contained as a child.
2. Select **Model | Add Diagram | Class Diagram** in the Menu Bar or select **Add Diagram | Class Diagram** in Context Menu.

Model Element is an abstract element of all UML model elements.

You can use **QuickEdit** for Model Element by double-click or press `Enter` on a selected model element.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

```
expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
stereotype ::= (identifier)
visibility ::= '+' | '#' | '-' | '~'
name ::= (identifier)
```

- **Visibility** : Change visibility property.
- **Add Note** : Add a linked note.
- **Add Constraint** : Add a constraint.

To create a Class:

1. Select **Class** in **Toolbox**.
2. Drag on the diagram as the size of Class.

To create a Class (model element only) by Menu:

1. Select an Element where a new Class to be contained.
2. Select **Model | Add | Class** in Menu Bar or **Add | Class** in Context Menu.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

  ```
  expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  ```

- **Visibility** : Change visibility property.
- **Add Note** : Add a linked note.
- **Add Constraint** : Add a constraint.
- **Add Attribute** (`Ctrl+Enter`) : Add an attribute.
- **Add Operation** (`Ctrl+Shift+Enter`) : Add an operation.
- **Add Template Parameter** : Add a template parameter.
- **Add Reception** : Add a reception.
- **Add Sub-Class** : Add a sub-class.
- **Add Super-Class** : Add a super class.
- **Add Provided Interface** : Add a provided interface.
- **Add Required Interface** : Add a required interface.
- **Add Associated Class** : Add an associated class.
- **Add Aggregated Class** : Add an aggregated class.
- **Add Composited Class** : Add a composited class.
- **Add Port** : Add a port.
- **Add Part** : Add a part.

To add an Attribute:

1. Select a Classifier.
2. Select **Model | Add | Attribute** in Menu Bar or **Add | Attribute** in Context Menu.

You can use **QuickEdit** for Attribute by double-click or press `Enter` on a selected Attribute.

- **Attribute Expression** : Edit Attribute expression.

  Syntax of Attribute Expression

  ```
  attribute ::= [ '<<' stereotype `>>` ] [ visibility ] name [':' type ] [ '[' multiplic
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  type ::= (identifier)
  multiplicity ::= multiplicity-bound [ '..' multiplicity-bound ]
  multiplicity-bound ::= (number) | '*'
  default-value ::= (string)
  ```

- **Visibility** : Change visibility property.
- **Add** (`Ctrl+Enter`) : Add one more attribute in the below.
- **Delete** (`Ctrl+Delete`) : Delete the attribute
- **Move Up** (`Ctrl+Up`) : Move the attribute up.
- **Move Down** (`Ctrl+Down`) : Move the attribute down.

To add an Operation:

1. Select a Classifier.
2. Select **Model | Add | Operation** in Menu Bar or **Add | Operation** in Context Menu.

You can use **QuickEdit** for Operation by double-click or press `Enter` on a selected Operation.

- **Operation Expression** : Edit Operation expression.

  Syntax of Operation Expression

  ```
  operation ::= [ '<<' stereotype `>>` ] [ visibility ] name [ '(' parameter-list ')' ]
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  parameter-list ::= parameter [ ',' parameter ]*
  parameter ::= (identifier)
  return-type ::= (identifier)
  ```

- **Visibility** : Change visibility property.
- **Add** (`Ctrl+Enter`) : Add one more operation in the below.
- **Delete** (`Ctrl+Delete`) : Delete the operation
- **Move Up** (`Ctrl+Up`) : Move the operation up.
- **Move Down** (`Ctrl+Down`) : Move the operation down.

To add a Template Parameter:

1. Select an Element.
2. Select **Model | Add | Template Parameter** in Menu Bar or **Add | Template Parameter** in Context Menu.

You can use **QuickEdit** for Template Parameter by double-click or press `Enter` on a selected Template Parameter.

- **Template Parameter Expression** : Edit Template Parameter expression.

  Syntax of Template Parameter Expression

  ```
  template-parameter ::= [ '<<' stereotype `>>` ] [ visibility ] name [':' type ] [ '='
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  type ::= (identifier)
  default-value ::= (string)
  ```

- **Visibility** : Change visibility property.
- **Add** (`Ctrl+Enter`) : Add one more template parameter in the below.
- **Delete** (`Ctrl+Delete`) : Delete the template parameter.
- **Move Up** (`Ctrl+Up`) : Move the template parameter up.
- **Move Down** (`Ctrl+Down`) : Move the template parameter down.

To create a Generalization:

1. Select **Generalization** in **Toolbox**.
2. Drag from an element (to be special) and drop on another element (to be general).

To create an Aggregation:

1. Select **Aggregation** in **Toolbox**.
2. Drag from an element (to be a part) and drop on another element (to be whole).

To create a Composition:

1. Select **Composition** in **Toolbox**.
2. Drag from an element (to be a part) and drop on another element (to be whole).

To create an Dependency:

1. Select **Dependency** in **Toolbox**.
2. Drag from an element (client) and drop on another element (supplier).

To create an Association (or Directed Association):

1. Select **Association** (or **Directed Association**) in **Toolbox**.
2. Drag from an element and drop on another element.

You can use **QuickEdit** for Relationship (See Relationship).

You can also use **QuickEdit** for Association End by double-click at the end side of an Association.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

  ```
  expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  ```

- **Visibility** : Change visibility property.
- **Navigability** : Change navigable property.
- **Aggregation Kind** : Change aggregationKind property.
- **Multiplicity** : Change multiplicity property.
- **Add Qualifier** : Add a qualifier (attribute) to the AssociationEnd.

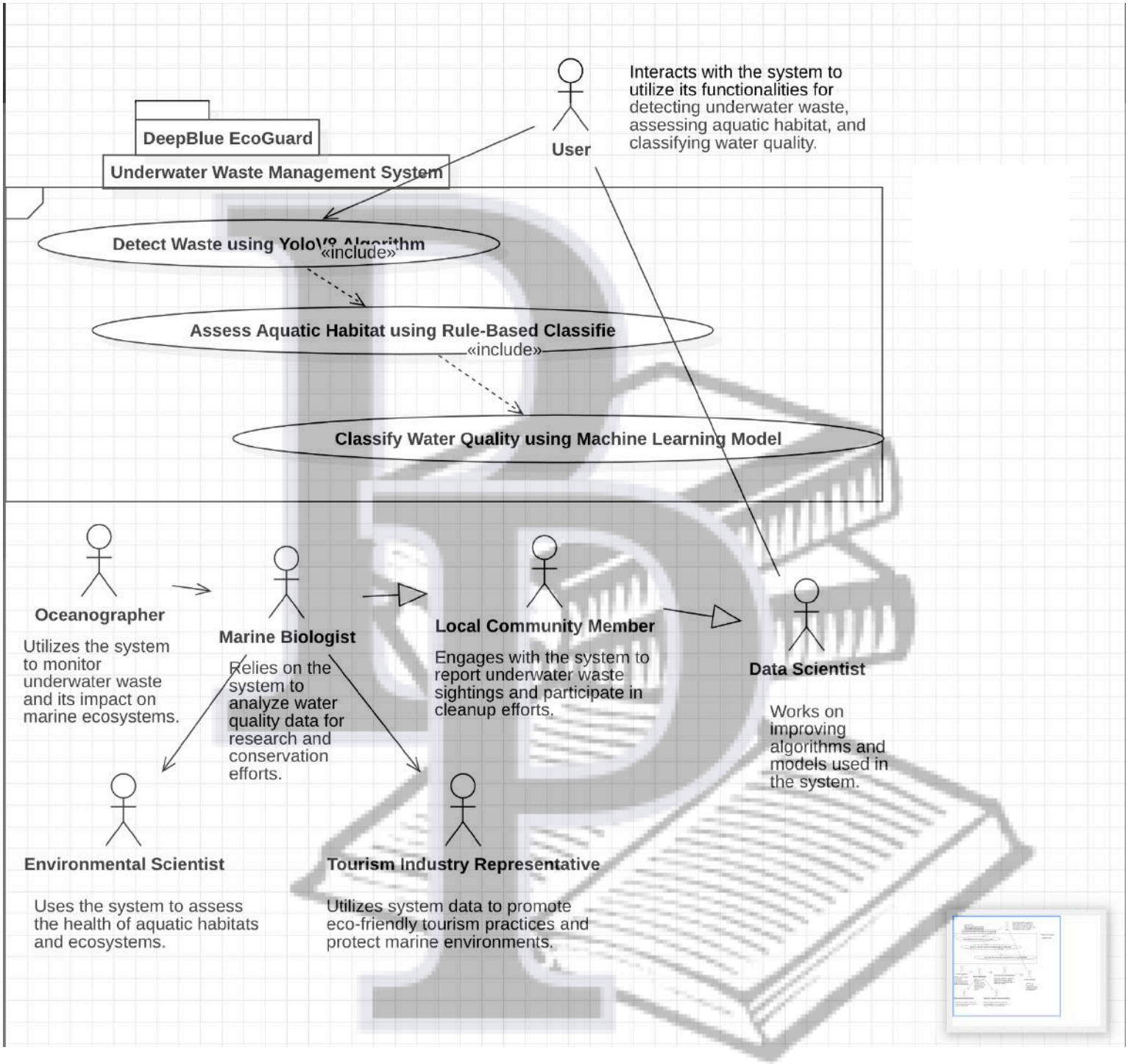
To add a Tag of an element:

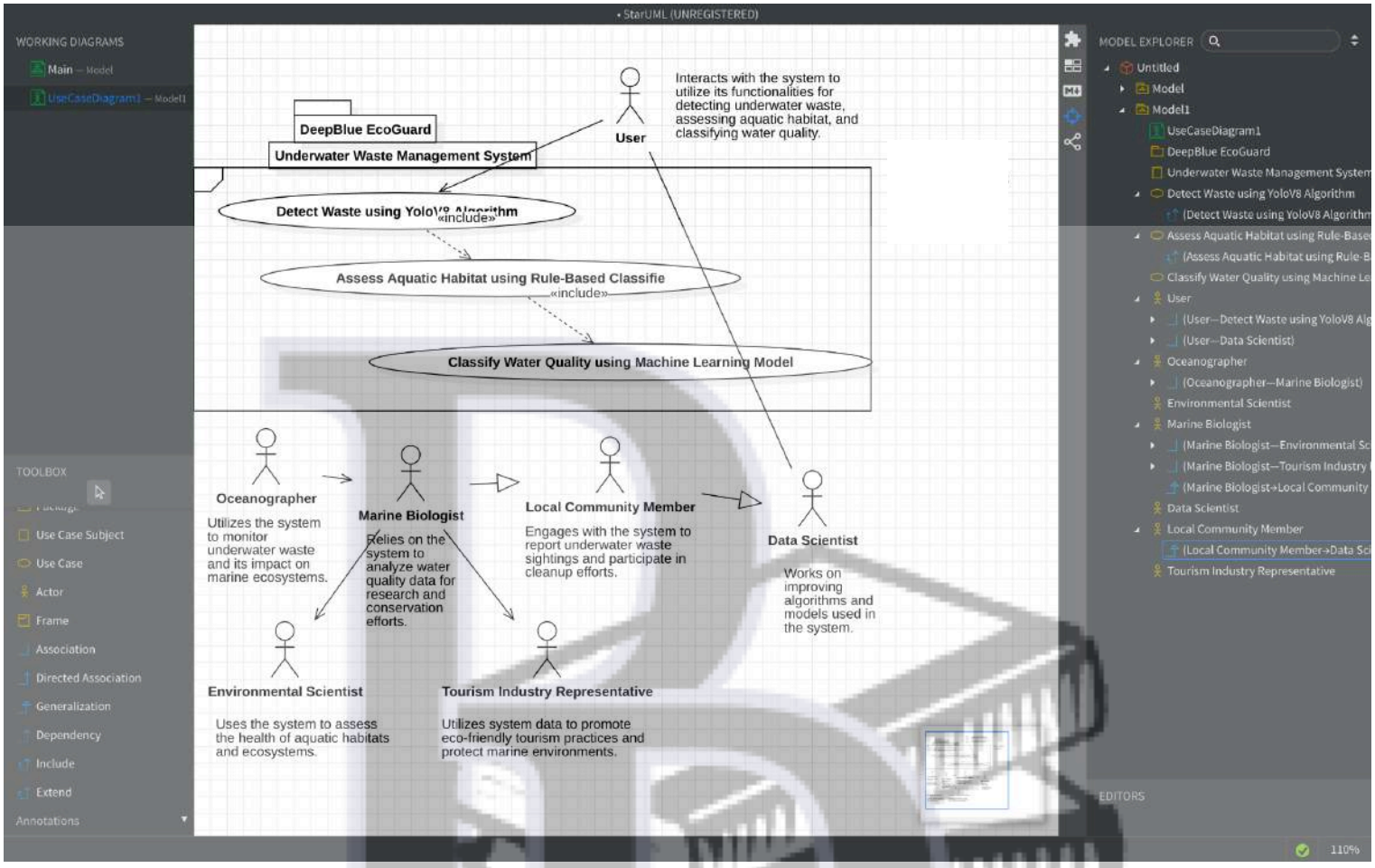
1. Select an element to add a Tag.
2. Select **Model | Add | Tag** in Menu Bar or **Add | Tag** in Context Menu.

You can select one of tag kinds: **string**, **reference**, **boolean**, or **number**. The `value` property corresponds to **string** kind. The `reference` property corresponds to **reference** kind. The `checked` property corresponds to **boolean** kind. The `number` property corresponds to **number** kind.

The Tags can be shown in views by checking **Format** > **Show Property** menu item. If you don't want to be shown, check `hidden` property.

3. Use Case Diagram





Package:
DeepBlue EcoGuard: Represents the overall system or project.

Use Case Subject:
Underwater Waste Management System: Represents the main subject of the use cases.

Use Cases:

- Detect Waste using YoloV8 Algorithm: Use case for detecting underwater waste using the YoloV8 algorithm.
- Assess Aquatic Habitat using Rule-Based Classifier: Use case for assessing aquatic habitat using a rule-based classifier.
- Classify Water Quality using Machine Learning Model: Use case for classifying water quality using a machine learning model.

Actors:

- Oceanographer: Utilizes the system to monitor and manage underwater waste.
- Environmental Scientist: Uses the system to assess aquatic habitat conditions.
- Marine Biologist: Relies on the system to analyze water quality for marine life conservation efforts.
- Government Environmental Agency: Monitors and regulates underwater waste management activities using the system.
- Underwater Drone Operator: Operates drones equipped with sensors to collect underwater data for the system.
- Data Scientist: Works on improving algorithms and models used in the system.
- Underwater Waste Cleanup Crew: Implements cleanup operations based on information provided by the system.
- Local Community: Engages with the system to report and address underwater waste issues in their area.

- Commercial Fisheries: Uses system insights to minimize the impact of underwater waste on fishing activities.
- Tourism Industry: Utilizes system data to promote eco-friendly tourism practices and protect marine ecosystems.
- Underwater Monitoring Station Operator: Manages monitoring stations collecting data for the system.

Frame:
Represents the boundary of the system and defines what is included within it. In this case, it would enclose the use cases and actors mentioned above.

Associations:
Connect use cases to actors to show which actors are involved in each use case.

Dependency:
Represents a relationship where one element depends on another, such as actors depending on the system to perform certain actions.

Generalization:
Indicates inheritance between actors, where a specialized actor inherits behaviors from a more general actor. For example, "Data Scientist" could be a specialization of "Computer Scientist".

This breakdown provides a comprehensive overview of the use case diagram elements and actors for the DeepBlue EcoGuard project.

CREATING USE CASE DIAGRAM USING STARUML:

To create a Use Case Diagram:

1. Select first an element where a new Use Case Diagram to be contained as a child.
2. Select **Model | Add Diagram | Use Case Diagram** in Menu Bar or select **Add Diagram | Use Case Diagram** in Context Menu.

To create an Use Case Subject:

1. Select **Use Case Subject** in **Toolbox**.
2. Drag on the diagram as the size of Use Case Subject.

To create an Actor:

1. Select **Actor** in **Toolbox**.
2. Drag on the diagram as the size of Actor.

To create an Actor (model element only) by Menu:

1. Select an Element where a new Actor to be contained.
2. Select **Model | Add | Actor** in Menu Bar or **Add | Actor** in Context Menu.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

  ```
  expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  ```

- **Visibility** : Change visibility property.
- **Add Note** : Add a linked note.
- **Add Constraint** : Add a constraint.
- **Add Attribute** (`Ctrl+Enter`) : Add an attribute.
- **Add Operation** (`Ctrl+Shift+Enter`) : Add an operation.
- **Add Sub-Actor** : Add a sub-actor.
- **Add Super-Actor** : Add a super actor.
- **Add Associated Use Case** : Add an associated use case.

To create an Use Case:

1. Select **Use Case** in **Toolbox**.
2. Drag on the diagram as the size of Use Case.

To create an Use Case (model element only) by Menu:

1. Select an Element where a new Use Case to be contained.
2. Select **Model | Add | Use Case** in Menu Bar or **Add | Use Case** in Context Menu.

To add an Extension Point:

1. Select an Use Case.
2. Select **Model | Add | Extension Point** in Menu Bar or **Add | Extension Point** in Context Menu.

You can use **QuickEdit** for Extension Point by double-click or press `Enter` on a selected Extension Point.

- **Name Expression** : Edit name expression.

  Syntax of Name Expression

  ```
  expression ::= [ '<<' stereotype `>>` ] [ visibility ] name
  stereotype ::= (identifier)
  visibility ::= '+' | '#' | '-' | '~'
  name ::= (identifier)
  ```

- **Visibility** : Change visibility property.
- **Add** (`Ctrl+Enter`) : Add one more extension point in the below.
- **Delete** (`Ctrl+Delete`) : Delete the extension point
- **Move Up** (`Ctrl+Up`) : Move the extension point up.
- **Move Down** (`Ctrl+Down`) : Move the extension point down.