Nour Jihene Agouf: Ph.D. Student

Company & Team Lab: Arolla & RMoD/Evref
Contact email: nour-jihene.agouf@inria.fr
Website: www.nouragouf.fr

Importance of Analysing and Maintaining the Quality of Software

Code is more than just functionality—it tells a story about design decisions and team collaboration.

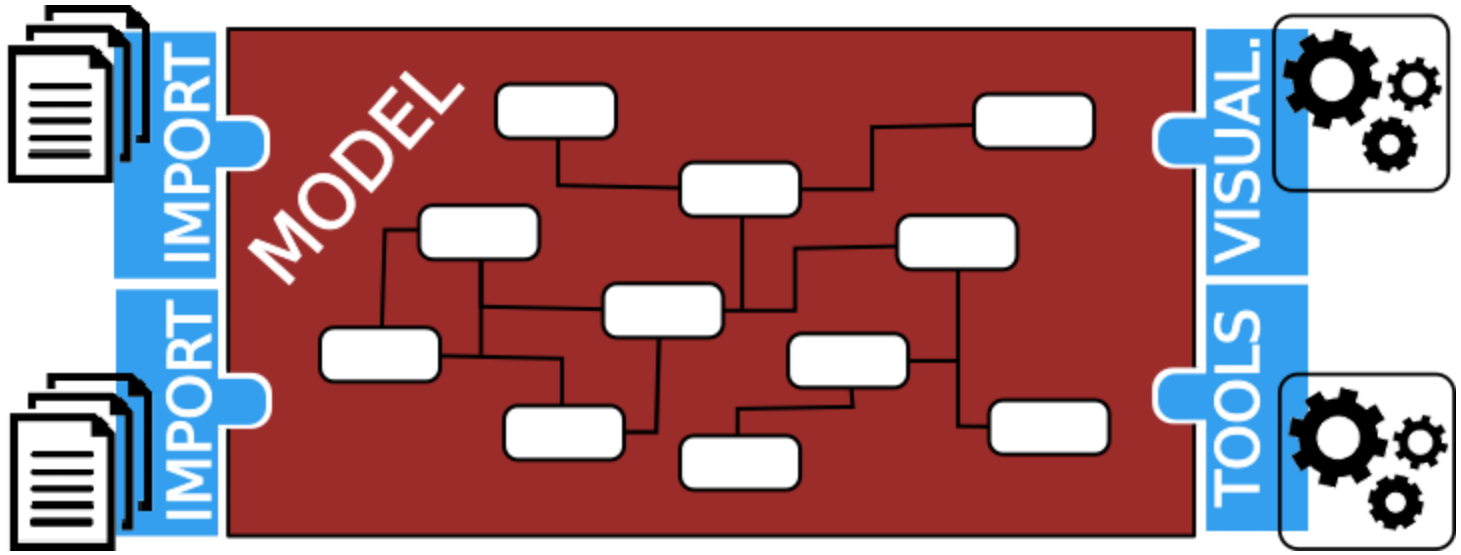
- Readability and Maintenance
- Reduces bugs and errors
- Improved performance
- Scalability
- Reusability
- Collaboration and team work

Moose

Moose is a plateform for the analysis and manipulation of software programs

It is scalable and designed to enable easy construction of new tools.

Moose



Components

- Metamodel Famix, software modelization (polyglote)
- Roassal, an engine for the creation of interactive visualizations
- MooseIDE, interactive environement of micro-tools on models
- *MooseQuery*, navigation and search inside of the model
- *Tagging*, semantic information such as *first class citizen*
- *Spec*, a bibliography for the construction of user interfaces

Moose Model and Metamodel: FAMIX

- FAMIX is a language-independent meta-model for representing software.
- Modelize any programming language such as Java, Pharo, Fortran, Ada, C/C++, TypeScript, Cobol, 4D.
- All Moose tools must function with these models of all languages.
- Based on traits (**composable** metamodel).

Pause ..

What are Traits?

Pause ..

What can we use if we want classes to share the same behavior?

Pause ..

- Abstract definition: It is a way between classes of distinct hierarchies to share the same behavior | it is a way to share behavior across classes without needing to use inheritance.
- Practical definition: It is a class that defines methods that could be shared between the classes who use it.

Moose Model and Metamodel: FAMIX

- Different entities: Package, Class, Method, Variable ...
- Their relationships (dependencies):
  - Inheritance `FamixJavaInheritance`
  - Access (to a variable) `FamixJavaAccess`
  - Invocations (of a method) `FamixJavaInvocation`
  - Reference (to a type) `FamixJavaReference`

Exercice

- Create a Moose Image
- Create a model of a Pharo project
- Navigate into the model

MooseQuery

- A uniformed query API of Famix entities
- Based on the description of the *Fame*
- Independent of the metamodel (of the modelized language)
- https://moosequery.ferlicot.fr/

Cheat sheet -- MooseQuery, parents/children

- `#children` (recursive `#allChildren`), ex: Package -> Package -> Class
- `#parents` (récursif `#allParents`), ex: Method -> Class -> Package
- Scopes: Search the ascendents or descendants with a given type
  - can "jump" levels: `methodeA atScope: FamixJavaPackage`
  - `#atScope: <Type>`, search ascendent (recursive `#allAtScope:`)
  - `#toScope: <Type>`, search descendent (recursive `#allToScope:`)

Cheat sheet -- MooseQuery, "neighbors"

- `#queryAllIncoming` / `#queryAllOutgoing` returns all the *associations* (FamixJavaInheritance, FamixJavaInvocation, ...)
  - Add `#opposites` to have the entities at the end of the associations
  - ex: `packageX queryAllIncoming opposites`
- `#query: <#in/#out> with: <association>`
  - ex: `methodA query: #in with: FamixTInvocation`
- query composition (all packages depending on *packageX*):
  `packageX queryAllIncoming opposites atScope: FamixTPackage`

Metamodel creation

- *Blog post* https://modularmoose.org/2021/02/15/Coasters.html
- Advice
  - Get inspired by an existing metamodel (ex: FamixJava)
  - Use at maximum the existing Famix traits
  - Do not begin until the metamodel is clear and complete
- creating a metamodel is a *long* task, after 20 years, FamixJava is yet not completely finished

VerveineJ

- Creation of a FamixJava model

  ```
  docker run -v "/local/source/dir":/src -v "/local/lib/dir":/dependency
  ghcr.io/evref-bl/verveinej:v3.0.7 -format json -o projet.json .
  ```

  - Produces a file `projet.json` containing the Famix model of the project
- Load the project in Moose (*ModelsBrowser* tool)
  - Attention to `rootFolder`

Exercice

- Clone this project: https://github.com/apache/maven
- Create the model of the project
- Import the project into Moose.
- Set the root folder of the project
- Navigate through its model to identify key classes or methods (For example: most references classes, most invoked methods).

Ressources

- https://modularmoose.org/
  - Wiki
  - Blog
- Github : https://github.com/moosetechnology
  - project MooseIDE
  - project Famix

Writing tests

Application

- Create a `MyChecker` class which has an attribute `model`.
  - create a method called `regexType:` which accepts a String and searched for the model classes with names that match the regex. Create a test for this method.

Application

- Create a method called `rootOf:` that accepts a class and returns the root class of the hierarchy to which it belongs.
  Write a test for this method.
- Create a method called `computeClassDepth:` that accepts a class as an argument and computes its depth in the hierarchy. Write a test for this method.
- Create a method called `computeHierarchyDepth:` that accepts a class as an argument and computes the depth of its whole hierarchy. Write a test for this method.

Application

- Create a method `isOverloading:` that checks weather the class is overloading (overloading means a class that has the same method with different arguments). Write a test for this method.
- Create a method `isBigClass:` that checks weather the class is big in terms of the number of methods it contains. Write a test for this method.

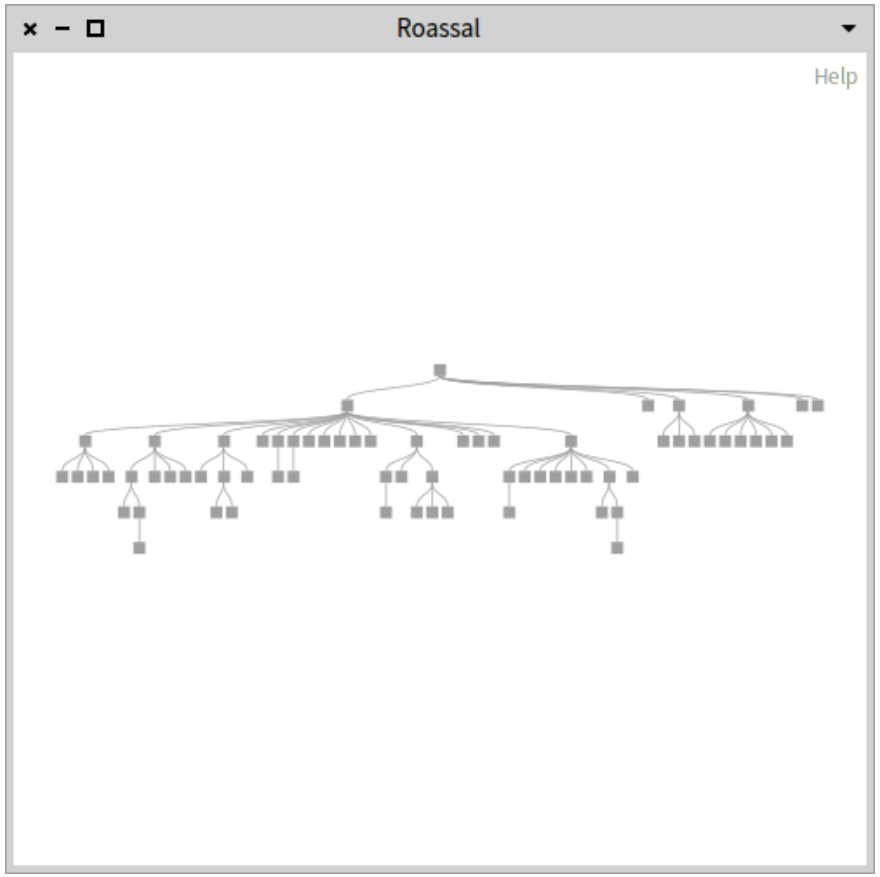
Back to Roassal

Application

For every class of the hierarchy `SequenceableCollection`, **collect** a rectangle that describes it.
The result must be a set of shapes.
Add these shapes to a canvas and open the canvas.

Application

Create links between classes and their superclass.
Display those classes to obtain a hierarchical view.



Application

Add events to each shape, allowing to inspect the model when a mouse click.

Application

Add an interaction that allows to display informations about classes when a mouse hover the object.
Informations :

- Name
- Number of methods
- Number of attributes
- Number of lines of code

Application

Edit the visualization to adapt to the size of classes according to these properties :

- Height : number of the methods of the class
- Width : number of attributes of the class
- Colour : number of lines of code

Making a user Interface using Spec

- For Pharo classes (classes in the Pharo environment)
- For classes in the Moose Model