Chapter 1

Analyzing Software using Moose

Moose is a platform for data and software analysis. Moose is the result of a collaborative effort, initiated at the University of Bern, which now comprise several compagnies and research groups spread all over the World. The main website of Moose is http://moosetechnology.org.

Moose is often used to craft specific software engineering tools, such tools guiding engineers to make proper decisions about the course of a software development. Humane Assessment is a method designed by Tudor Gîba and largely described on the website http://www.humane-assessment.com.

What Moose is good for: Moose is made to carry out analysis on the source code structure. A whole range of analysis may be conducted by analysis source code. Moose is also frequently used to analyze configuration files, process description, mailing list.

Data processing may be automatized. Embedding Pharo within a batch job is easy and is a practical monitoring solution.

What Moose is not good for: Moose cannot directly be used to analyse the dynamic execution of an application. Dynamic analysis consists in recording relevant information during a program execution. Such information may then be used to spot particular behavior.

1.1 Moose in a nutshell

The core of Moose is composed of:

- *Panel*: An extensible tool to navigate, visualize, and formulate queries.

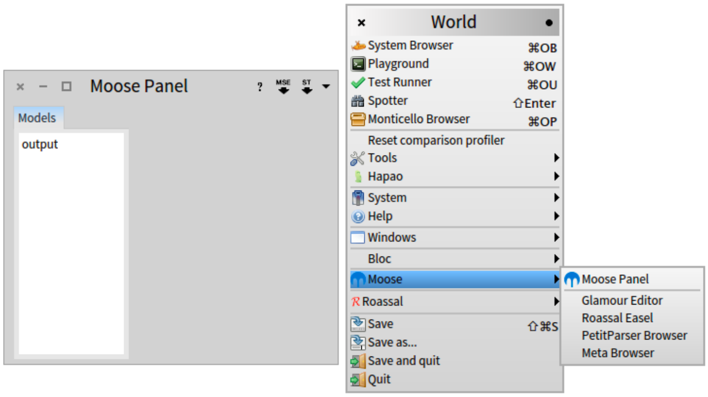


Figure 1.1: Opening the Moose Panel.

- *Parsers*: Two parsers are shipped with Moose. MSE is a file-format describe source code models. MSE is typically produced by externally provided parsers.
- *The Famix metamodel*: An abstract representation of source code. Famix is generic and it may describes application written in Java, C, C#, and Pharo.
- *Chef*: A domain-specific language to query structural dependencies. Analysis the dependencies between software elements is particularly difficult. Chef allows one to precisely formulate queries

1.2 Loading a software application

Creating a model in Moose is probably the very first step to take. Such model will then define the Universe in which your analyse will be carried out. The standard way to create a model is to load an application. To do so, the Moose panel has to be open. The panel is accessible from the World menu (Figure 1.1).

Moose has two built-in parsers: one for Pharo, and another for MSE. The first parser is accessible by clicking on the **ST** icon, top-right of the Moose Panel. Clicking on it will open a package selector. You simply have to select the Pharo packages you wish to includes in your model. The second parser takes as input a MSE file, accessible by clicking on the **MSE** button.

MSE is a file format that describe source code. Consider the following Java code:

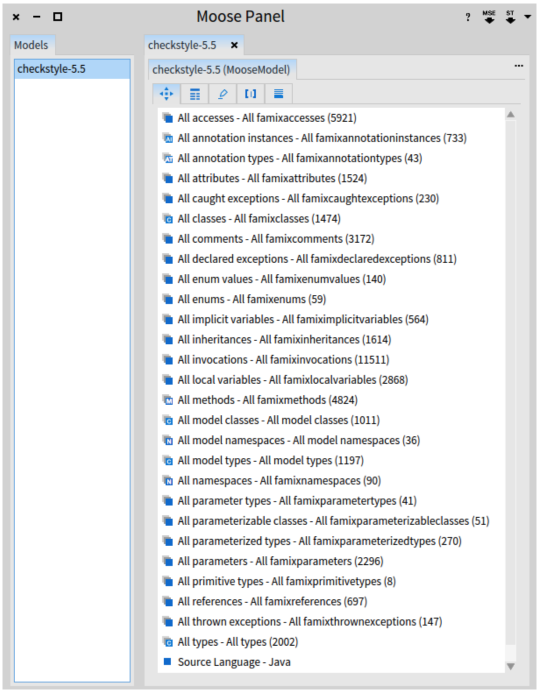
```
package agilevisualization.example;

public class HelloWorld {
   public static void main(String[] argv) {
   System.out.println("Hello World");
   }
}
```

The corresponding MSE file of that code is long of 48 lines. The first few lines are:

```
(
   (FAMIX.Inheritance (id: 3)
      (subclass (ref: 10))
      (superclass (ref: 4)))
   (FAMIX.Class (id: 4)
      (name 'Object')
      (container (ref: 7))
      (isStub true))
   (FAMIX.FileAnchor (id: 5)
      (element (ref: 1))
      (endLine 6)
      (fileName './agilevisualization/example/HelloWorld.java')
      (startLine 4))
   (FAMIX.FileAnchor (id: 6)
      (element (ref: 10))
      (endLine 7)
      (fileName './agilevisualization/example/HelloWorld.java')
      (startLine 3))
   ...
)
```

Producing a MSE file from Java source code is done using VerveineJ, Unfortunately, at the time this chapter is being written, VerveineJ has a proprietary licence. Contact the book author or the Moose mailing list for more information.

Along this chapter, we will use a running example, obtained from the Java application checkstyle. Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. The source code of Checkstyle is available from its website: http://checkstyle.sourceforge.net. We have produced a MSE file, available on http://bit.ly/checkstyle.



All accesses - All famixaccesses (5921)
All annotation instances - All famixannotationinstances (733)
All annotation types - All famixannotationtypes (43)
All attributes - All famixattributes (1524)
All caught exceptions - All famixcaughtexceptions (230)
All classes - All famixclasses (1474)
All comments - All famixcomments (3172)
All declared exceptions - All famixdeclaredexceptions (811)
All enum values - All famixenumvalues (140)
All enums - All famixenums (59)
All implicit variables - All famiximplicitvariables (564)
All inheritances - All famixinheritances (1614)
All invocations - All famixinvocations (11511)
All local variables - All famixlocalvariables (2868)
All methods - All famixmethods (4824)
All model classes - All model classes (1011)
All model namespaces - All model namespaces (36)
All model types - All model types (1197)
All namespaces - All famixnamespaces (90)
All parameter types - All famixparametertypes (41)
All parameterizable classes - All famixparameterizableclasses (51)
All parameterized types - All famixparameterizedtypes (270)
All parameters - All famixparameters (2296)
All primitive types - All famixprimitivetypes (8)
All references - All famixreferences (697)
All thrown exceptions - All famixthrownexceptions (147)
All types - All types (2002)
Source Language - Java

Figure 1.2: Loading checkstyle.

1.3 Browsing the source code

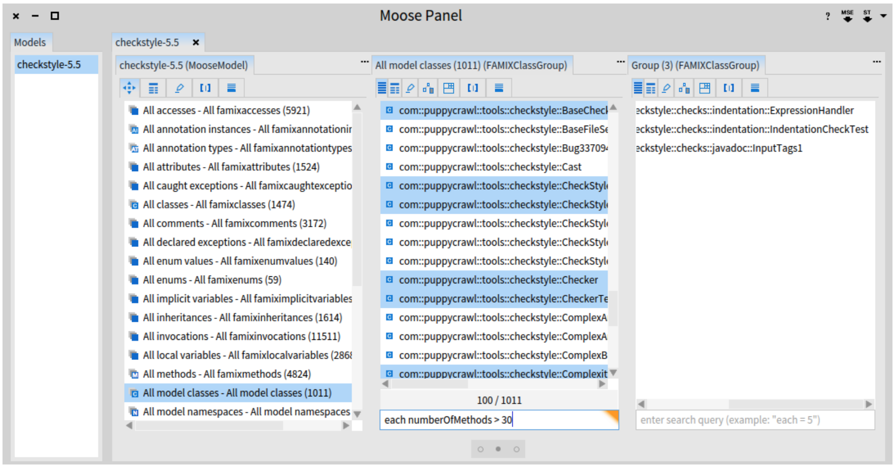
After having loaded the checkstyle MSE file, a new model is listed in the Moose (Figure 1.2).

Selecting the checkstyle model in the panel gives you a long list of kind of entities presents in the model. The panel indicates a few interesting facts about Checkstyle. In particular, Checkstyle is composed of

- 4,824 methods (item *All methods*)
- 1,011 classes (item *All model classes*)
- 36 packages (item *All model namespaces*)

The item *All classes* lists all the classes defined in the application and all the classes used by checkstyle that are not define in it. Example of such classes are Object, String, Integer, which are defined in the Java runtime.

Select the *All model classes* item to list all the classes defined in the applications. Below the list of classes, a text input accepts filtering queries. Enter


Moose Panel

Models
checkstyle-5.5

checkstyle-5.5 (MooseModel)
All accesses - All famixaccesses (5921)
All annotation instances - All famixannotationir
All annotation types - All famixannotationtypes
All attributes - All famixattributes (1524)
All caught exceptions - All famixcaughtexceptio
All classes - All famixclasses (1474)
All comments - All famixcomments (3172)
All declared exceptions - All famixdeclaredexce
All enum values - All famixenumvalues (140)
All enums - All famixenums (59)
All implicit variables - All famiximplicitvariables
All inheritances - All famixinheritances (1614)
All invocations - All famixinvocations (11511)
All local variables - All famixlocalvariables (286
All methods - All famixmethods (4824)
All model classes - All model classes (1011)
All model namespaces - All model namespaces

All model classes (1011) (FAMIXClassGroup)
com::puppycrawl::tools::checkstyle::BaseChecl
com::puppycrawl::tools::checkstyle::BaseFileSe
com::puppycrawl::tools::checkstyle::Bug33709
com::puppycrawl::tools::checkstyle::Cast
com::puppycrawl::tools::checkstyle::CheckStyl
com::puppycrawl::tools::checkstyle::CheckStyl
com::puppycrawl::tools::checkstyle::CheckStyl
com::puppycrawl::tools::checkstyle::CheckStyl
com::puppycrawl::tools::checkstyle::CheckStyl
com::puppycrawl::tools::checkstyle::Checker
com::puppycrawl::tools::checkstyle::CheckerTe
com::puppycrawl::tools::checkstyle::ComplexA
com::puppycrawl::tools::checkstyle::ComplexA
com::puppycrawl::tools::checkstyle::ComplexB
com::puppycrawl::tools::checkstyle::Complexit
100 / 1011
each numberOfMethods > 30

Group (3) (FAMIXClassGroup)
eckstyle::checks::indentation::ExpressionHandler
eckstyle::checks::indentation::IndentationCheckTest
eckstyle::checks::javadoc::InputTags1
enter search query (example: "each = 5")

Figure 1.3: Querying in the panel.

each numberOfMethods > 30. A third pane appears and list the classes defined in checkstyle that have more than 30 methods (Figure 1.3).

Since the item displayed in the panel *All model classes* are instances of the class FAMIXClass, the each variable refers to an FAMIXClass. Browsing this class using the Pharo code browser will give you the list of methods that are understandable by the each variable.

Queries involing the class name are also often formulated. For example, the query

```
'*Log*' match: each name
```

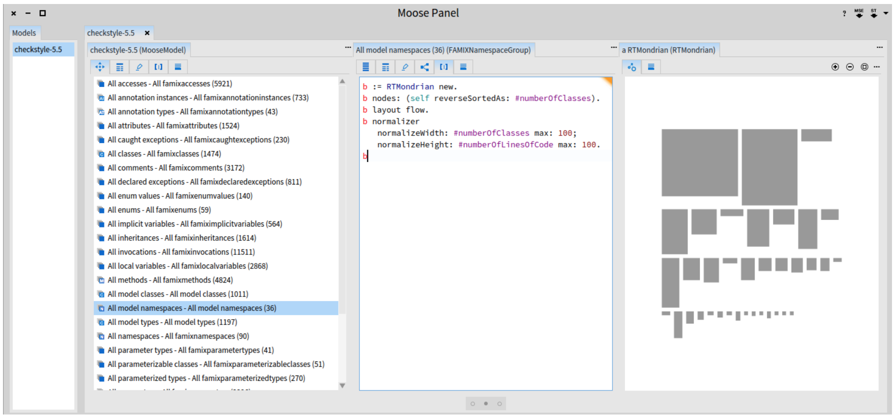
designates classes defined in Checkstyle that have the word Log in their name. The string being an expression regular, matched using match:.

Another interesting query is each isStub (you need to select the item *All classes* instead). This simple query list only the classes that are considered as stub. In a Moose jargon, a stub class is a class that is used by the analyzed application but not defined in it. Checkstyle contains 398 stub classes, corresponding to classes defined in other libraries, not part of the .MSE file. For example, Checkstyle depends on some libraries provided by the Apache foundation or Google. These libraries are not part of the .MSE file.

Query may be relatively complex. For example, consider:

```
each numberOfMethodsOverriden > (each numberOfMethodsInherited / 3)
```

This query returns the classes that override more than 1/3 of the classes


Moose Panel
Models
checkstyle-5.5
checkstyle-5.5
checkstyle-5.5 (MooseModel)
All model namespaces (36) (FAMIXNamespaceGroup)
a RTMondrian (RTMondrian)

All accesses - All famixaccesses (5921)
All annotation instances - All famixannotationinstances (733)
All annotation types - All famixannotationtypes (43)
All attributes - All famixattributes (1524)
All caught exceptions - All famixcaughtexceptions (230)
All classes - All famixclasses (1474)
All comments - All famixcomments (3172)
All declared exceptions - All famixdeclaredexceptions (811)
All enum values - All famixenumvalues (140)
All enums - All famixenums (59)
All implicit variables - All famiximplicitvariables (564)
All inheritances - All famixinheritances (1614)
All invocations - All famixinvocations (11511)
All local variables - All famixlocalvariables (2868)
All methods - All famixmethods (4824)
All model classes - All model classes (1011)
All model namespaces - All model namespaces (36)
All model types - All model types (1197)
All namespaces - All famixnamespaces (90)
All parameter types - All famixparametertypes (41)
All parameterizable classes - All famixparameterizableclasses (51)
All parameterized types - All famixparameterizedtypes (270)

```
b := RTMondrian new.
b nodes: (self reverseSortedAs: #numberOfClasses).
b layout flow.
b normalizer
  normalizeWidth: #numberOfClasses max: 100;
  normalizeHeight: #numberOfLinesOfCode max: 100.
b
```

Figure 1.4: Querying in the panel.

that it inherits. Such classes are rather suspicious since it may be worth considering whether they belongs to the adequate class hierarchy.

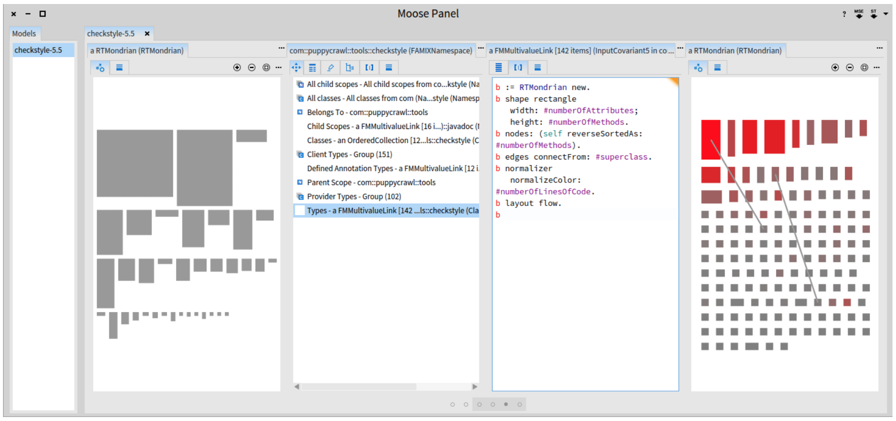
1.4 Visualizing the code distribution

Checkstyle contains more than one thousand classes, distributed over 36 packages. Understanding the code distribution over these packages may involve a dedicated visualization. After having selected the checkstyle model, select the packages by pressing *All model namespaces*. The new pane located on the right-hand side lists these packages. Oddly enough, the packages <Default Package> and blah are listed.

Pressing the [|] tab on the right pane allows one to enter a script. The self variable is then bound to the collection of packages. Enter the script (Figure 1.4):

```
b := RTMondrian new.
b nodes: (self reverseSortedAs: #numberOfClasses).
b layout flow.
b normalizer
   normalizeWidth: #numberOfClasses max: 100;
   normalizeHeight: #numberOfLinesOfCode max: 100.
b
```

This simple visualization shows that not all the packages contains the same amount of code. The first package, the largest, is com.puppycrawl.tools.checkstyle. Clicking on the package shows up a new


Moose Panel
Models
checkstyle-5.5
checkstyle-5.5
a RTMondrian (RTMondrian)
com::puppycrawl::tools::checkstyle (FAMIXNamespace)
a FMMultivalueLink [142 items] (InputCovariant5 in co...
a RTMondrian (RTMondrian)

All child scopes - All child scopes from co...kstyle (Na
All classes - All classes from com (Na...style (Namesp
Belongs To - com::puppycrawl::tools
Child Scopes - a FMMultivalueLink [16 i...)::javadoc (
Classes - an OrderedCollection [12...ls::checkstyle (C
Client Types - Group (151)
Defined Annotation Types - a FMMultivalueLink [12 i.
Parent Scope - com::puppycrawl::tools
Provider Types - Group (102)
Types - a FMMultivalueLink [142 ...ls::checkstyle (Cla

```
b := RTMondrian new.
b shape rectangle
   width: #numberOfAttributes;
   height: #numberOfMethods.
b nodes: (self reverseSortedAs:
#numberOfMethods).
b edges connectFrom: #superclass.
b normalizer
   normalizeColor:
#numberOfLinesOfCode.
b layout flow.
b
```

Figure 1.5: Classes in the largest package.

pane. The second tab gives the list of properties for that package. In indicates that it defines 142 classes, spread over 5,617 lines of code.

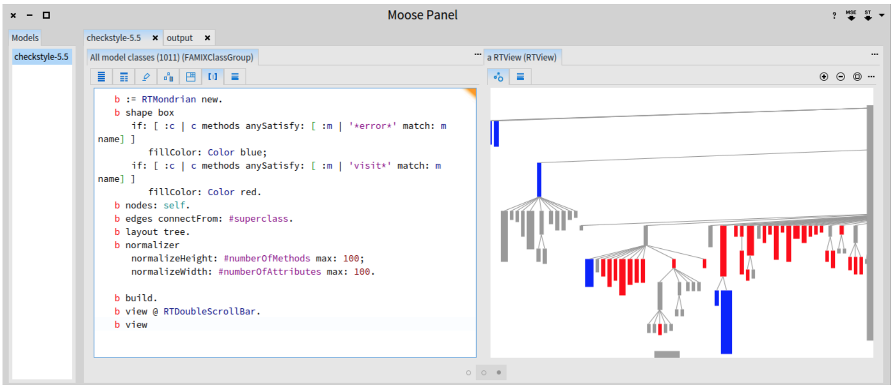
Clicking on the entry *Types* lists all the classes defined in the package checkstyle. The following script reveals a number of facts regarding these 142 classes (Figure 1.5):

```
b := RTMondrian new.
b shape rectangle
   width: #numberOfAttributes;
   height: #numberOfMethods.
b nodes: (self reverseSortedAs: #numberOfMethods).
b edges connectFrom: #superclass.
b normalizer
   normalizeColor: #numberOfLinesOfCode.
b layout flow.
b
```

Most of the classes are small, in terms of lines of code, number of defines methods and attributes. Moreover, only two of these classes are subclasses.

1.5 Visualizing cross-cutting concerns

Cross-cutting features are features that crosscut software structural entities, such as classes or packages. Due to their nature, cross-cutting are not explicitely modularized, and may therefore lead to some code anomalies. Consider the following script executed on the list of model classes (Figure 1.6):



```
  b := RTMondrian new.
  b shape box
     if: [ :c | c methods anySatisfy: [ :m | '*error*' match: m
name] ]
        fillColor: Color blue;
     if: [ :c | c methods anySatisfy: [ :m | 'visit*' match: m
name] ]
        fillColor: Color red.
  b nodes: self.
  b edges connectFrom: #superclass.
  b layout tree.
  b normalizer
     normalizeHeight: #numberOfMethods max: 100;
     normalizeWidth: #numberOfAttributes max: 100.

  b build.
  b view @ RTDoubleScrollBar.
  b view
```

Figure 1.6: Visualizing cross-cutting concerns.

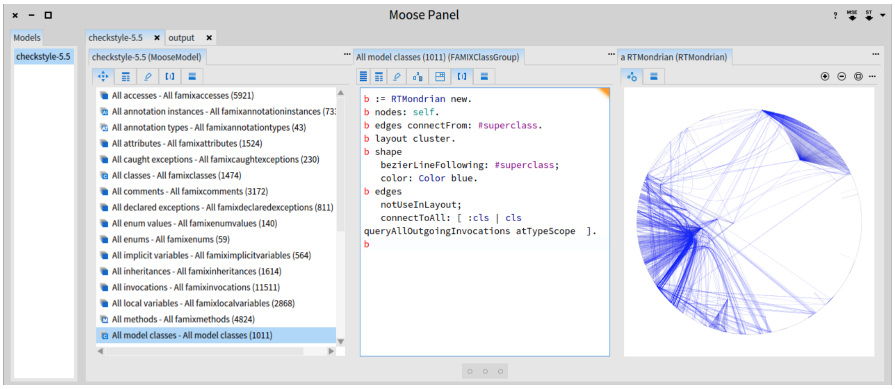
```
b := RTMondrian new.
b shape box
   if: [ :c | c methods anySatisfy: [ :m | '*error*' match: m name ] ]
      fillColor: Color blue;
   if: [ :c | c methods anySatisfy: [ :m | 'visit*' match: m name ] ]
      fillColor: Color red.
b nodes: self.
b edges connectFrom: #superclass.
b layout tree.
b normalizer
   normalizeHeight: #numberOfMethods max: 100;
   normalizeWidth: #numberOfAttributes max: 100.

b build.
b view @ RTDoubleScrollBar.
b view
```

The script paints in blue classes that have at least one method with the word error in it and in red classes that are visiting some entities. The hierarchy of red classes indicates a specialization of visitors. It is also interesting to notice that errors are so rarely handled.

1.6 Dependencies using Moose Chef

Moose chef is an API allowing to query dependencies between packages, classes, and methods. Consider the following script, running on the list of model classes (Figure 1.7):

```
b := RTMondrian new.
```


Moose Panel
Models
checkstyle-5.5
checkstyle-5.5
output
checkstyle-5.5 (MooseModel)
All model classes (1011) (FAMIXClassGroup)
a RTMondrian (RTMondrian)

All accesses - All famixaccesses (5921)
All annotation instances - All famixannotationinstances (73
All annotation types - All famixannotationtypes (43)
All attributes - All famixattributes (1524)
All caught exceptions - All famixcaughtexceptions (230)
All classes - All famixclasses (1474)
All comments - All famixcomments (3172)
All declared exceptions - All famixdeclaredexceptions (811)
All enum values - All famixenumvalues (140)
All enums - All famixenums (59)
All implicit variables - All famiximplicitvariables (564)
All inheritances - All famixinheritances (1614)
All invocations - All famixinvocations (11511)
All local variables - All famixlocalvariables (2868)
All methods - All famixmethods (4824)
All model classes - All model classes (1011)

```
b := RTMondrian new.
b nodes: self.
b edges connectFrom: #superclass.
b layout cluster.
b shape
   bezierLineFollowing: #superclass;
   color: Color blue.
b edges
   notUseInLayout;
   connectToAll: [ :cls | cls
queryAllOutgoingInvocations atTypeScope  ].
b
```

Figure 1.7: Dependencies between classes.

```
b nodes: self.
b edges connectFrom: #superclass.
b layout cluster.
b shape
   bezierLineFollowing: #superclass;
   color: Color blue.
b edges
   notUseInLayout;
   connectToAll: [ :cls | cls queryAllOutgoingInvocations atTypeScope ].
b
```

Sending the message queryAllOutgoingInvocations to a Famix class returns a list of Famix invocations. An invocation is an instance of the Famix class FAMIXInvocation. Invocations represents a low level abstract that often need to be transformed. Such transformation is called setting a scope by Moose Chef. The invocations may be scoped to the classes by sending atTypeScope. For a given Famix class cls, the expression cls queryAllOutgoingInvocations atTypeScope returns the list of Famix classes that cls depends on. A Chef query may be scoped using atNamespaceScope, atPackageScope, atTypeScope, atMethodScope.

1.7 More on that topic

Moose is a large project that contains many exciting facets. This chapter only superficially highlights some of them. Another good source of documentation is the Moose Book (http://www.themoosebook.org/book), project leaded by Dr. Tudor Gîrba http://www.tudorgirba.com.

Numerous research articles have been using Moose. Many of them are accessible from:

- http://rmod.inria.fr/web/publications
- http://scg.unibe.ch/publications/scg-pub