A Metrics Front-End for the Moose Reengineering Environment

Informatikprojekt

vorgelegt von

Calogero Butera

2002

Leiter der Arbeit:

Prof. Dr. Oscar Nierstrasz
Prof. Dr. Stéphane Ducasse
Michele Lanza

Institut für Informatik und angewandte Mathematik
Universiät Bern

Abstract

Moose is a language independent tool environment to reverse engineer and reengineer object-oriented systems. It consists of a repository to store models of software systems, provides query and navigation facilities, metrics and other analysis support. Models consist of entities representing software artifacts such as classes and methods.

This document describes a metrics front-end for Moose, whose goal is to visualize the relationship among different metrics of the same model and export this information to an external file.

With this tool the user can analyse software metrics and observe the relationships among them. Collecting these informations leads to a better understanding of the software which has to be analysed or reengineered. The ability to export metric values into an external statistic tool such as MS Excel enables the user to create diagrams and apply statistical analysis methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 MOOSE

The *Moose Reengineering Environment* is a language independent tool environment to reverse engineer, understand, and reengineer software systems. Moose supports reengineering of applications developed in different object-oriented languages. It supports reengineering by providing facilities for analyzing and storing multiple models, for refactoring and by providing support for analysis methods such as metrics and the inference of properties of source code elements [DLT01].

With its fully object-oriented implementation, Moose provides a complete description of the metamodel elements in terms of objects that are easily parameterized, extended or manipulated. These properties make Moose an ideal foundation for reengineering tools such as CodeCrawler [DDL99] or Supremo [KN01].

1.2 Metrics and Entities

Moose uses entities and metrics as an important source of information for the analysis of the software. Entities build a central role in the Moose environment. They represent software artifacts such as classes, methods, etc.

Metrics measure certain properties of a software project by mapping them to numbers (or other symbols) according to well-defined, objective measurement rules. The measurement results are then used to describe, judge or predict characteristics of a software project. Thus software metrics support numerous reengineering tasks, because they help to focus reengineering efforts. They aid in forming an initial understanding of the legacy system and can often uncover hints about design flaws that are obstructing the modification and the extension of the system.

Metrics of software artifacts can be categorized in different groups such as *Complexity Metrics*, *Coupling Metrics*, *Cohesion Metrics* and *Inheritance Tree Metrics* [DD99].

1.3 Motivation of the Project

Moose is a powerful tool which simplifies the analysis and the understanding of a software which has to be reengineered. At the same time Moose has a lot of potential extensions. However a feature that is not provided through a friendly user interface can become difficult to use. The need of some user-friendly interfaces becomes stronger and stronger. Through such interfaces many existing features of a software can be used in a different manner. This makes many functionalities more accessible and interesting to the user.

Momfe and its Components In the context of a student project we built Momfe (a Moose Metrics Front-end). Momfe can be described as a group of graphical user interfaces and functionalities which use existing features of Moose to visualize the meaning of any software metrics, configure a metrics table by choosing the wanted metrics and export the configured table to an external file which can be imported in MS Excel or any other statistic software. This gives the user the opportunity to analyze the chosen metric values and make statistical evaluation of that data.

1.4 Structure of this Document

This document is structured in three parts. In order to describe Momfe and its features, we begin with an introduction (chapter 1) of Moose explaining the concept of metrics and their importance. The presentation of Momfe is done in the second part (chapter 3). Here we explain what Momfe is and its features. To show the potential of Momfe we do an evaluation of Momfe in the third part. Here we show what can be done with Momfe and what its utility is. In the appendix A we speak about the architecture of Momfe and with a short user manual we explain how to use it.

Chapter 2

The Moose Reengineering Environment

2.1 Moose

The Moose Reengineering Environment. In the context of the FAMIX project, a Reengineering Environment called Moose has been developed at the IAM of the University of Berne. This language independent tool environment was created with the goal to simplify the reverse engineering and reengineering of complex software systems. Moose was implemented in VisualWorks (a Smalltalk programming environment). It consists of a repository to store models of software systems, and provides facilities to analyze, query and navigate them. Models consist of entities representing software artifacts such as classes, methods, etc.

Moose, an ideal Foundation for Reengineering Tools. The software engineer can load the software he is going to analyze into the Moose environment, which creates a model. In order to work with this model it is necessary to divide it in different elements which should be able to provide some information to the user. The fully object-oriented implementation of Moose provides a complete description of the meta model elements in terms of objects that are easily parameterized, extended or manipulated [DLT01]. These properties make Moose an ideal foundation for reengineering tools such as CodeCrawler [DDL99] or Supremo [KN01].

2.2 Entities

Entities build a central role in the MOOSE environment. They represent software artifacts such as classes, methods, etc. Every entity is represented by an object, which allows direct interaction and consequently an easy way to query and navigate a whole model.

2.3 Metrics

Metrics measure certain properties of a software project by mapping them to numbers (or other symbols) according to well-defined, objective measurement rules. The measurement results are then used to describe, judge or predict characteristics of a software project with respect to the property that has been measured. Usually, measurements are made to provide a foundation of information upon which decisions about software engineering tasks can be both planned and performed better [DD99].

2.3.1 The importance of Metrics

Software Metrics support numerous reengineering tasks, because they help to focus reengineering efforts. They aid in forming an initial understanding of the legacy system and can often uncover hints about design flaws that are obstructing the modification and the extension of the system. Metrics lend themselves to automation and with appropriate tools they can provide easy access to meaningful information about the source code without requiring the software engineer to read through all the source code by hand. Instead you can use the information to make a more efficient study of the source code based on the points of interest indicated by the metrics result.

2.3.2 Categories of Software metrics

Software Metrics fall into several categories depending on the aspects of a system they measure. We can identify the following categories: *Complexity Metrics*, *Coupling Metrics*, *Cohesion Metrics* and *Inheritance Tree Metrics* [DD99]. The table 2.1 shows some examples of software metrics.

Name	Description
HNL	*Hierarchy nesting level*, also called *depth of inheritance tree*. The number of classes in superclass chain of class. In case of multiple inheritance, count the number of classes in the longest chain.
NA	*Number of accessors*, the number of get/set - methods in a class.
NAM	*Number of abstract methods*.
NC	*Number of constructors*.
NCV	*Number of class variables*.
NIA	*Number of inherited attributes*, the number of attributes defined in all superclasses of the subject class.
NIV	*Number of instance variables*.
NMA	*Number of methods added*, the number of methods defined in the subject class but not in its superclass.
NME	*Number of methods extended*, the number of methods redefined in subject class by invoking the same method on a superclass.
NMI	*Number of methods inherited*, i.e. defined in superclass and inherited unmodified.
NMO	*Number of methods overridden*, i.e. redefined in subject class.
NOC	*Number of immediate children of a class*.
NOM	*Number of methods*, each method counts as 1. NOM = NMA + NME + NMO.
NOMP	*Number of method protocols*. This is Smalltalk - specific: methods can be grouped into method protocols.
PriA	*Number of private attributes*.
PriM	*Number of private methods*.
ProA	*Number of protected attributes.*
ProM	*Number of protected methods*.
PubA	*Number of public attributes*.
PubM	*Number of public methods*.
WLOC	*Lines of code*, sum of all lines of code in all method bodies of the class.
WMSG	*Number of message sends*, sum of number of message sends in all method bodies of class.
WMCX	*Sum of method complexities*.
WNAA	*Number of times all attributes defined in the class are accessed.*
WNI	*Number of method invocations*, i.e. in all method bodies of all methods.
WNMAA	*Number of all accesses on attributes*.
WNOC	*Number of all descendants*, i.e. sum of all direct and indirect children of a class.
WNOS	*Number of statements*, sum of statements in all method bodies of class.

Table 2.1: The class metrics used in this project.

Chapter 3

Momfe, the Moose Metrics Front-End

Momfe. Momfe is a tool of Moose, which was developed in the context of a student project at the IAM (Institute of Computer Science and Applied Mathematics) of the University of Berne. The major goal of the project was to provide a graphical user interface (GUI) to visualize information about metrics, represent the metric values in a table and export this data into an external file, which can be loaded into MS Excel or any other statistical tool. The last feature leads to new possibilities for the analysis of the source code of the software by displaying diagrams or computing statistical evaluations.

3.1 Momfe and its Components

Momfe consists of six components:

- The Moose Model Metrics Viewer
- The Metrics Descriptor
- The Metrics Manager
- The Metrics Table Configurator
- The Metrics Table
- The Metrics Table Exporter

In this chapter we describe all these components from the user point of view. We will also show some examples with some figures in order to give an idea of the appearance of Momfe to the reader.

3.2 The Moose Model Metrics Viewer

The *Moose Model Metrics Viewer* was implemented in order to give a first overview on a model loaded in Moose. This helps the user to get an idea of the size and the complexity of the model he is going to analyze. It gives also an idea of the effort needed to analyze, extend, reengineer or simply understand the source code.

The GUI of the *Moose Model Metrics Viewer* is implemented in such a manner that the user sees in a first overview information about the system he is going to analyze. Thus the Moose Model Metrics Viewer displays a GUI containing a table which shows the number of:

- Classes
- Methods
- Attributes
- Inheritance Definitions
- Invocations
- Accesses

If the user needs more informations, he can click on the button *"Show Complete Table"* and some supplementary data will be displayed.

- Access Arguments
- Entities
- Expression Arguments
- Formal Parameters
- Global Variables
- Implicit Variables
- Named Properties

The examples in the figures 3.1 and 3.2 show the values for the *LanApp*, an example Model.

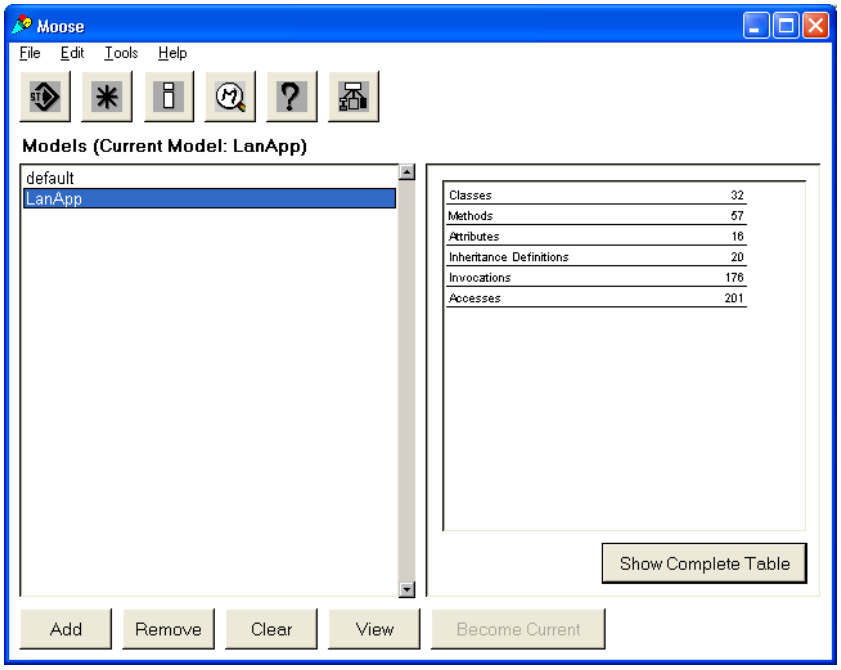

Figure 3.1: Moose Model Metrics Viewer. The partial overview of the LanApp.

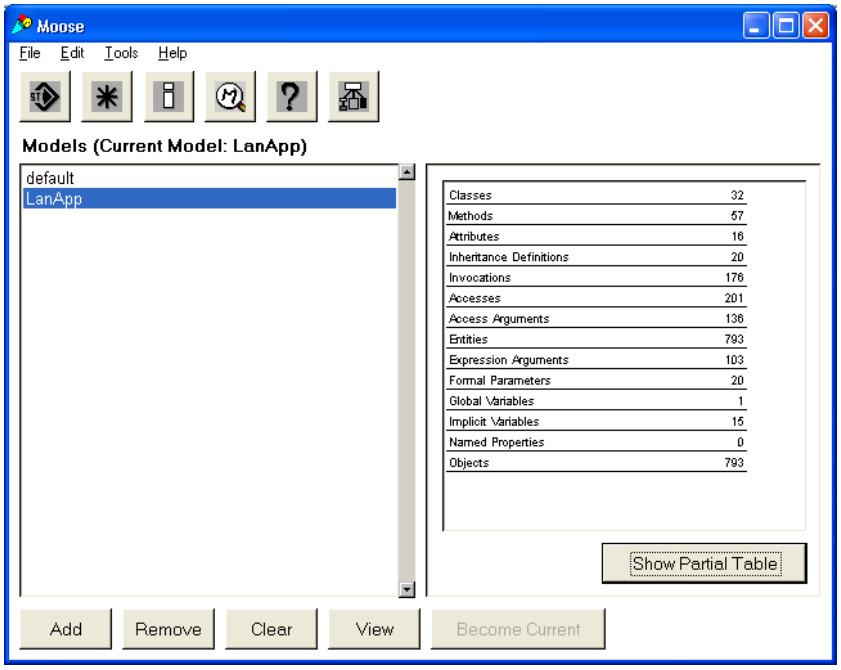

Figure 3.2: Moose Model Metrics Viewer. The complete overview of the LanApp.

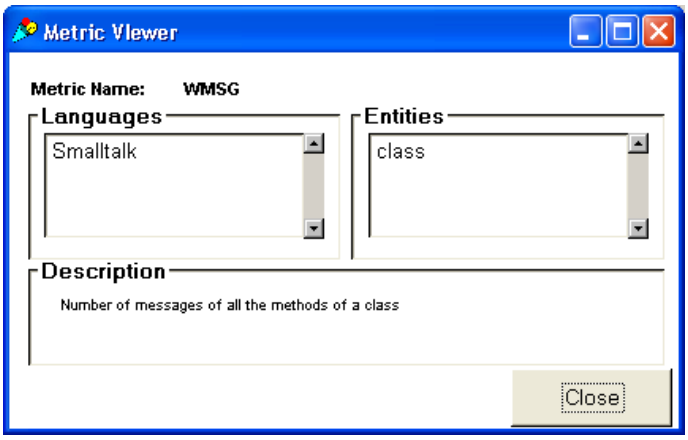

Figure 3.3: The Metric Descriptor. Displaying information about the WMSG-metric. WMSG is the *number of messages of all the methods of a class*, WMSG can be applied on classes. This value is only available for models of software written in Smalltalk.

3.3 The Metric Descriptor

One little feature which can be helpful for the Moose user is the *Metric Descriptor*. In Moose metrics are often displayed as acronyms. As an example we can take the *WMSG*-metric. Let us suppose that the user sees this acronym and he is not confident with the Moose system. He will have some questions about WMSG like: What does this acronym mean? On which kind of entities is it applied and can it be applied on the source code written in a specific language?

The Metric Descriptor can be considered as the solution to this problem. It consists of a GUI which displays the following information about a chosen metric:

- A short description of the metric
- The metric name (acronym)
- The languages supported by this metric
- The entities on which the metric is applied

The figure 3.3 represents an example where the user can see the description of *WMSG*. WMSG is the *number of messages of all the methods of a class*, WMSG can be applied on classes. This value is only available for models of software written in Smalltalk.

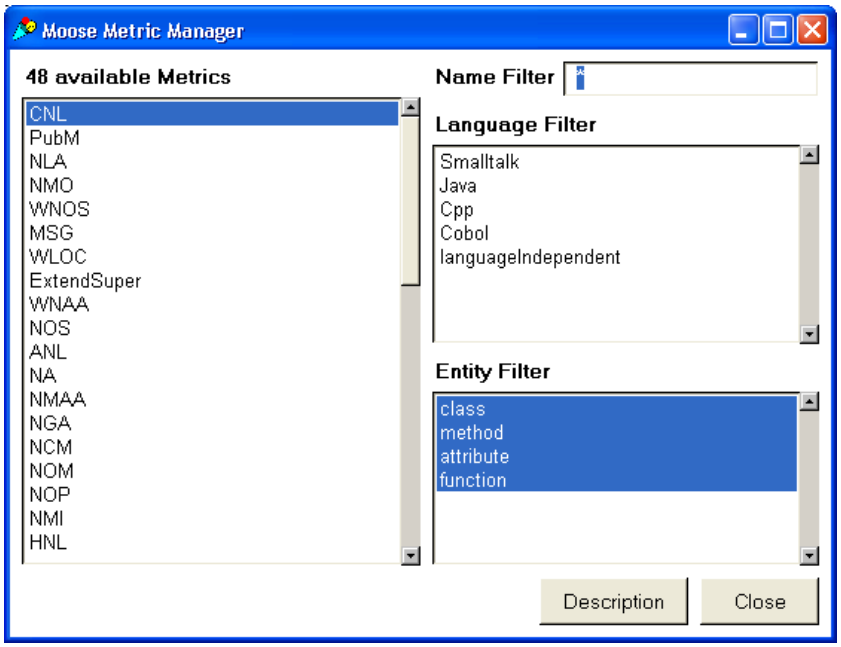

Figure 3.4: The Moose Metric Manager. Without any filtering it shows 48 different metrics.

3.4 The Metric Manager

The *Metric Descriptor* without any tool which provides the possibility to find the desired metric does not make any sense. This observation led us to the implementation of a new tool, the *Metric Manager*. This tool should give an overview on how many and which metrics the Moose system provides and the possibility to select any of them and get a description of it.

The GUI of the *Moose Metric Manager* enables the user to look for a metric using different criteria. The searching criteria are: the language, the entity, and a string filter which enables the user to find the metric he is looking for. The string filter supports wild-cards. Double-clicking on an metric launches the *Metric Descriptor* and the description of the chosen metric is given. The GUI also contains a button called *"Description"* which offers the same functionality.

In the figures 3.4, 3.5, 3.6, 3.7 and 3.8 the reader can see some snapshots of the *Moose Metric Manager*. Figure 3.4 shows the GUI without selecting any criteria, in figure 3.5 we used the language filter, in figure 3.6 we used the entity filter and in figure 3.7 the name filter was applied. As already mentioned the user can get the description of any already selected metric (like in figure 3.8) at any time.

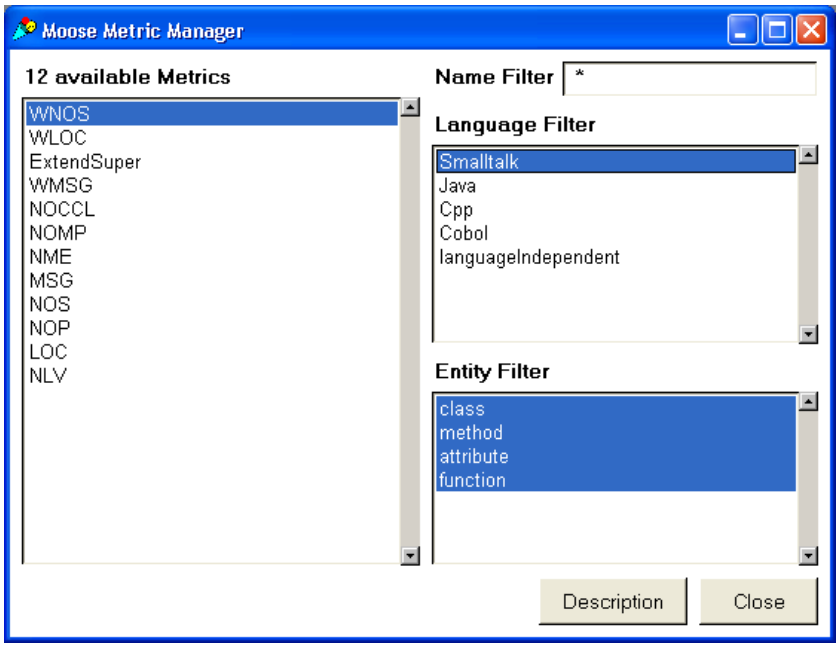

Figure 3.5: The Moose Metric Manager. There are 12 different metrics that can be used on source code written in Smalltalk.

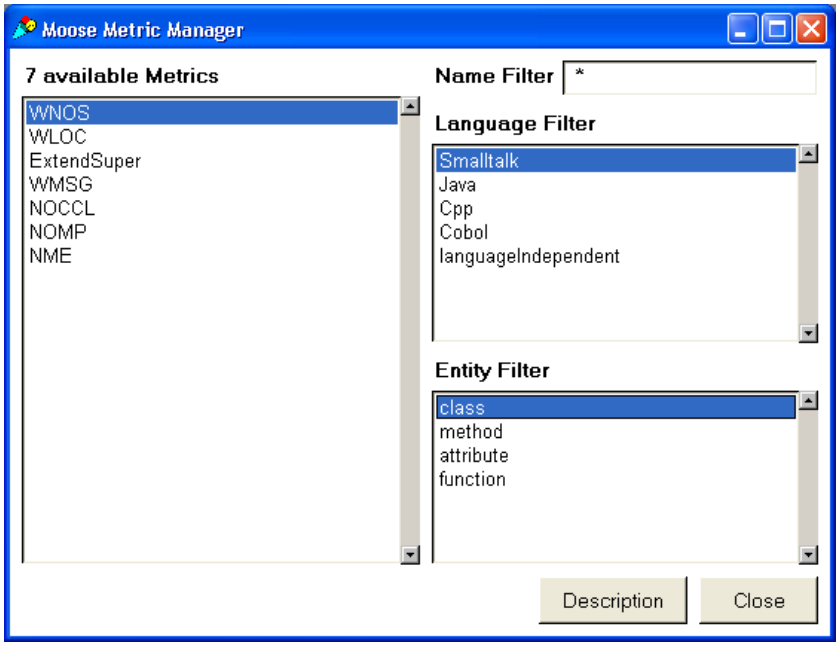

Figure 3.6: The Moose Metric Manager. There are 7 different class metrics that can be used on source code written in Smalltalk.

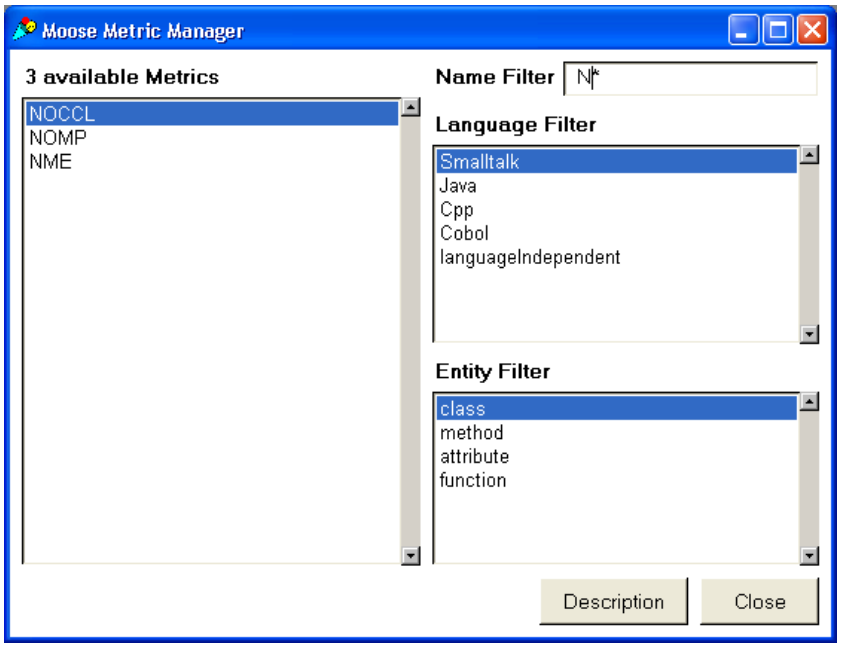

Figure 3.7: The Moose Metric Manager. There are 3 different class metrics beginning with the letter N that can be used on source code written in Smalltalk.

3.4.1 The Metrics Table Configurator

The principal goal of the project is to display a table with the metric values for a chosen set of entities. This feature should be configurable. That means that the user must be able to choose the metrics that match the following criteria.

- The metrics can be applied to a specific group of entities (classes, attributes, methods, functions) chosen by the user
- The metrics work only for a specific programming language, are language-independent or both
- The metric's acronyms match to a string. The use should be able to use wild-cards. (Example: "*", "N*", "N*A" etc.)

The *Metrics Table Configurator* helps the user to choose the metric values that will be displayed in the columns of the *Entity Metric Table* (*EMT*). As soon as the selection of the metrics is done, the user will click on the button *"Open Table"*. The result of this action is that the configurator window is closed and a new window contains the *Entity Metric Table* appears.

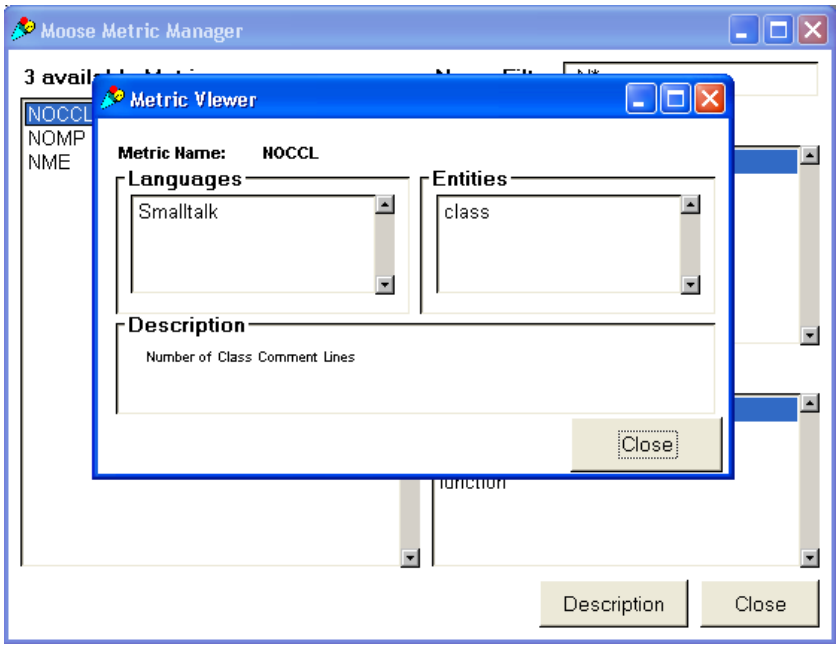

Figure 3.8: The Moose Metric Manager. The description of the NOCCL metric.

The figures 3.9, 3.10 and 3.11 show what the *Metrics Table Configurator* looks like and some examples of how the filtering fields could be used.

3.4.2 The Entity Metric Table (EMT)

The *Entity Metric Table (EMT)* shows for each chosen entity kind and metric the related metric values. The rows represent the entities and the columns the metrics. It is possible to sort the values by double-clicking on the column titles. If the table does not correspond to the user's wish it is possible to reconfigure it by clicking on the Configuration button. This GUI representing the *Entity Metric Table (EMT)* gives also the possibility to export the table into an external file. This file can be imported into MS-Excel in order to make statistical analysis on the collected results. That task is provided by the button "Export to Excel". Figure 3.12 shows the EMT for the LOC and NOS metrics applied to the methods of the example application.

3.4.3 The Excel Exporter

The *Table Exporter* is the last component of the Momfe project. Its task is to take the data collected in the EMT and export them to an external (tabulator-separated) file.

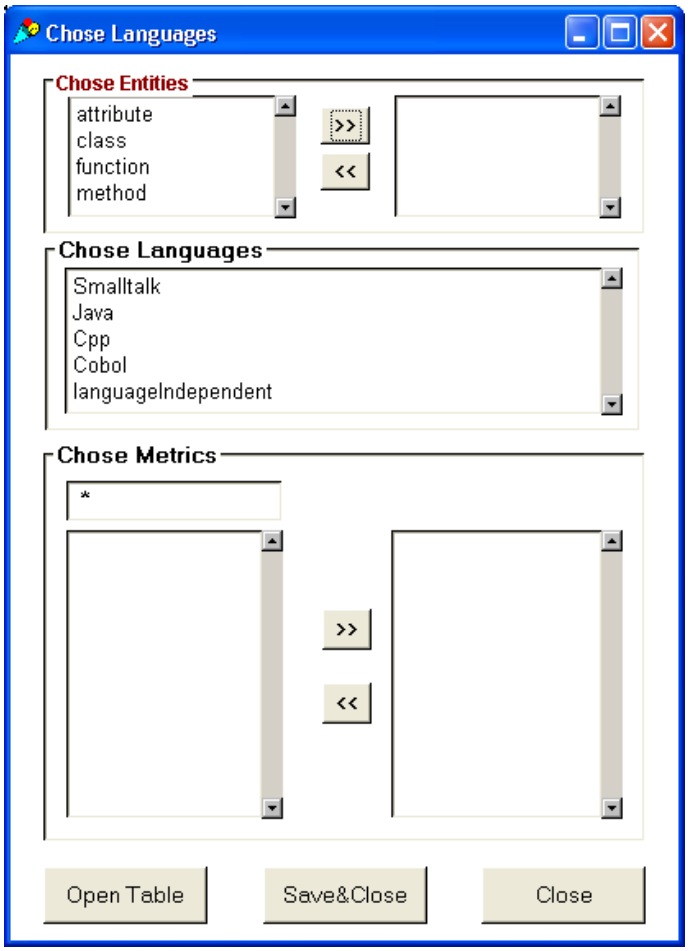

Figure 3.9: The Moose Metric Table Configurator. The user can chose the metrics matching few filter criteria.

When the user clicks on the *"Export to Excel"* button a dialog appears. This dialog asks the user for the path and filename of the file, which is going to be created. If the user only gives the filename, the file is saved in the working directory. The application just takes the values displayed in the table and writes a stream and saves it into the file. Figure 3.13 shows the dialog that appears after clicking the button *"Export to Excel"*.

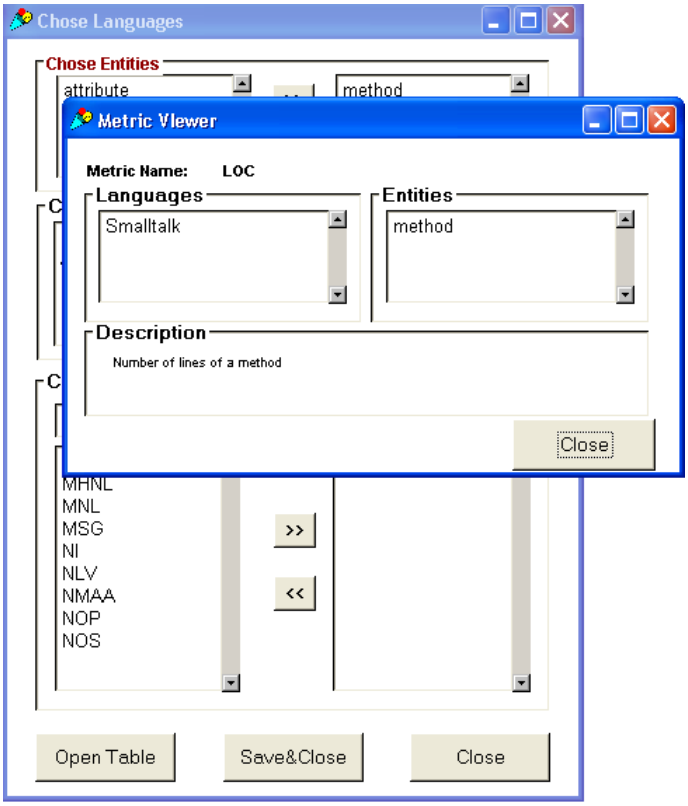

Figure 3.10: The Moose Metric Table Configurator. Like in the Metric Manager, the user can double-click on a metric to know more about the chosen metric.

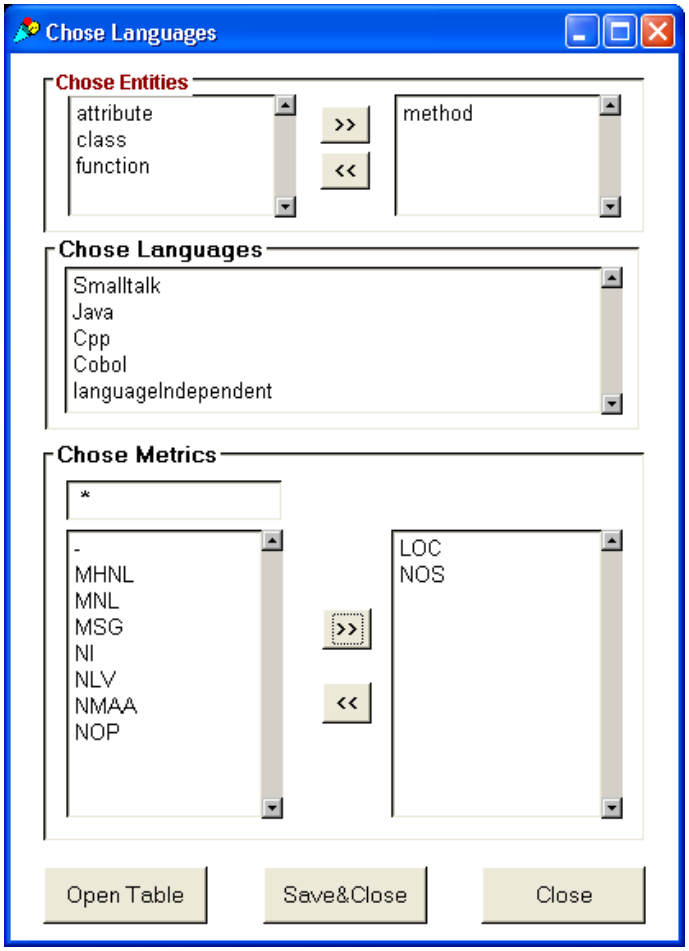

Figure 3.11: The Moose Metric Table Configurator. If the user clicks on ”Open Table”, an EMT appears with the LOC and the NOS values for the entity kind methods.

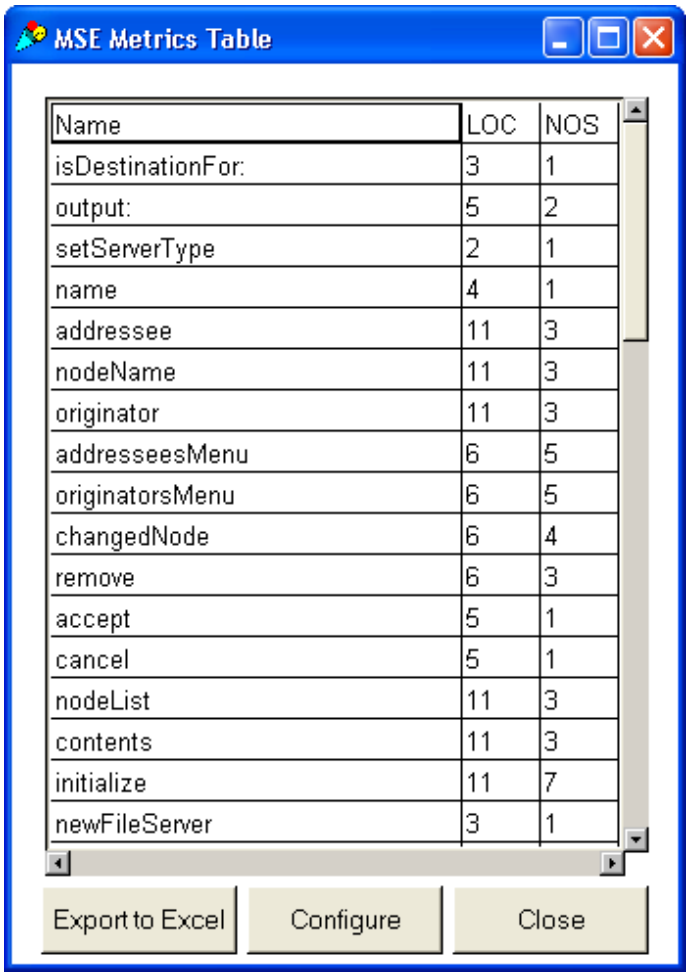

Figure 3.12: The Entity Metric Table. The table can be reconfigured following the requests of the user.

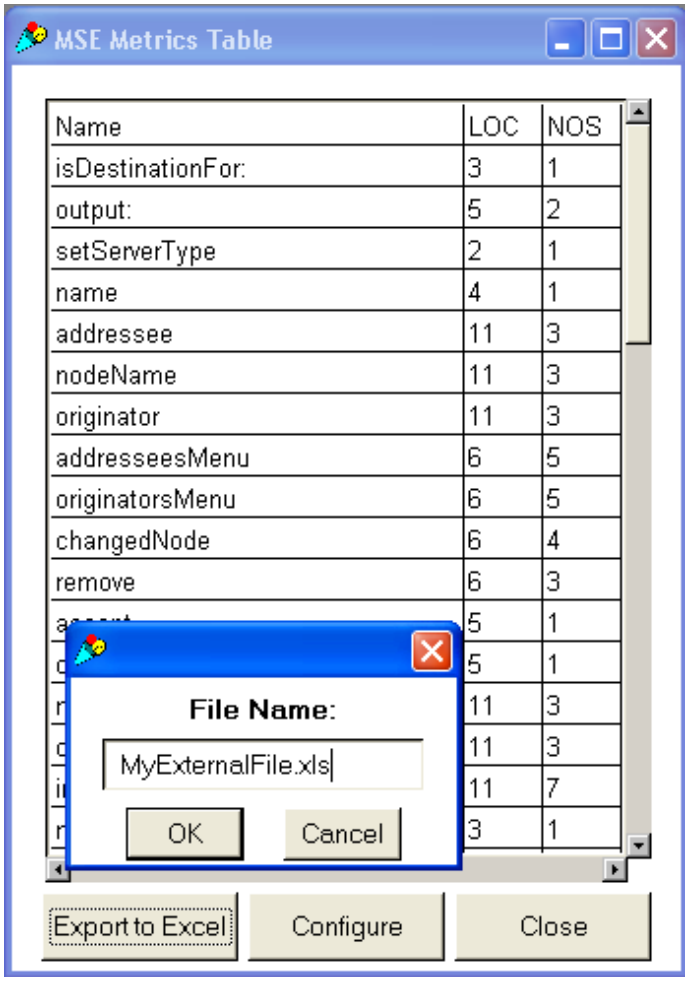

Figure 3.13: The Entity Metric Table. After clicking the button ”Export to Excel” a dialog asks the user to enter a path and a filename.

Chapter 4

Evaluation

4.1 Introduction

Moose is a powerful system which helps the software engineer to analyze the source code of a software in order to improve, modify or simply understand it. Momfe provides a user-friendly interface to some functionalities and extends them with an export tool. In this chapter we will present the utility of Momfe working on an case study.

4.2 Case Study

Before doing the case study we have to decide what model we want to chose, what entities we are interested in, and which metrics we want to visualize in order to analyze them later using a tool like MS Excel. In this section we go through all important steps of our case study: The choice of a specific case, the data collection with Momfe, and the evaluation of the results.

4.2.1 Presentation of the Case Study

We chose the Refactoring Browser Version 3.5.1 written in Smalltalk for the VisualWorks 3.0 environment as the software code to analyze. We decided to choose some metrics which can be computed on the entity kind *methods*. The following list presents the 7 metrics that we chose:

- **LOC** *Number of lines of a method*
- **MHNL** *Depth of a method within the inheritance hierarchy*
- **MNL** *Method Name Length*
- **MSG** *Number of messages of a method*
- **NI** *Number of method invocations within a method*

- **NMAA** *Number of attribute accesses within a method*
- **NOS** *Number of statements of a method*

After loading the model of the Refactoring Browser into Moose we already have a first view on the model and we see that this software contains 248 classes, 719 methods, 4060 invocations, 17621entities etc. (see figure 4.1).

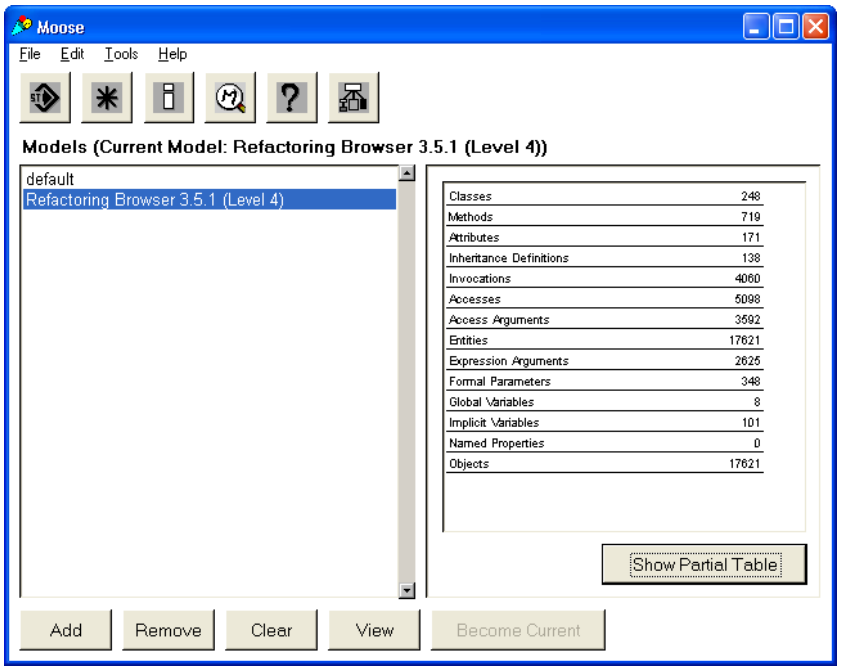

Figure 4.1: Here we see a first overview on the system we want to deal with (the Refactoring Browser V 3.5.1.

After loading the model we must run the Moose operators in order to compute the values for the metrics which are not yet stored in the Moose model (see figure 4.2). The operators we run on this case study are the following:

- *Candidate Invocations With Base Operator*
- *Language Independent Metrics Operator*
- *Smalltalk Annotation Operator*
- *Smalltalk Candidate Invocation Operator*
- *Smalltalk Metrics Operator*

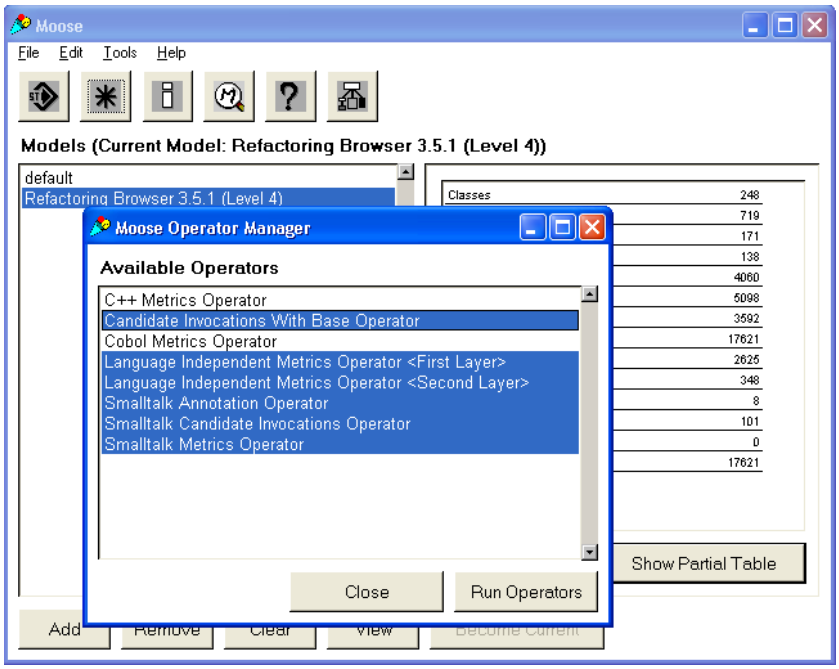

Figure 4.2: To get missing metric values, we run these operators on the loaded model.

The next steps is the configuration of the Entity-Metric-Table and its visualization. Later we export these data and load them into MS Excel where we make some statistical evaluation and visualize the data with some diagrams.

4.2.2 Doing the case study

To study the values of the metrics we are interested in, we run the *Metric Table Configurator*, we choose the LOC, MHNL, MNL, MSG, NI, NMAA and the NOS metric, then we open the *EMT* where we can already see our values (see figure 4.4). To do some evaluation in MS Excel we need to export the data into a file.

Now we do some statistical analysis. We computed the following parameters:

- the mean value (see table 4.1)
- the median (see table 4.2)
- the variance (see table 4.3)
- the standard deviation (see table 4.4)
- the correlation between each pair of metrics (see tables 4.5 and 4.6)

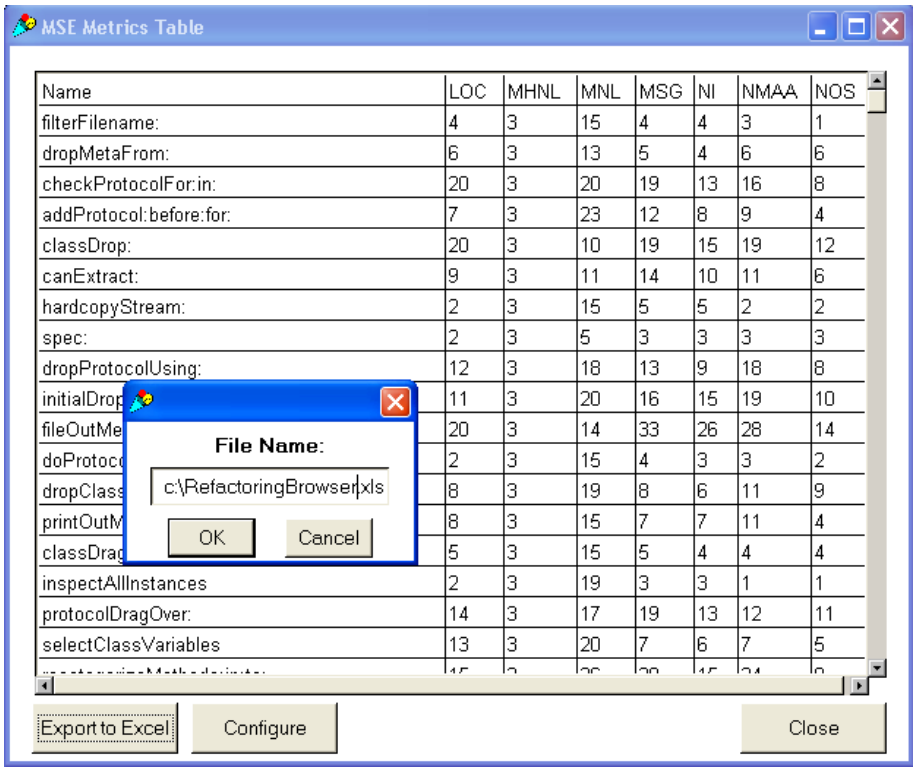

Figure 4.3: The Entity Metric Table for the Refactoring Browser V 3.5.1.

4.2.3 Case Study Evaluation

After loading the model into Moose, choosing the metrics, doing some statistical evaluation we can observe the following. The methods of the *Refactoring Browser V 3.5.1* have on average about seven lines of code, the method name length is on average about nine character, and the methods contain on average about three invocations. We also observe that 50% of the methods do only one invocation. We can also look at the correlation matrices which tell us that each of the MSG, NI, NMAA and NOS metric do correlate strongly with the other three (These metrics are defined in the table 2.1. That means that:

- ***NI** correlates strongly with NOS, NMAA and MSG*
- ***NMAA** correlates strongly with MSG, NI and NOS.*
- ***NOS** correlates strongly with MSG, NI and NMAA*
- ***MSG** correlates strongly with NI, NMAA and NOS*

Before doing this case study we could expect this relationships otherwise we should think about refactoring the software to make it easier to understand and to

LOC	7,44
MHNL	1,35
MNL	9,35
MSG	3,72
NI	3,09
NMAA	3,53
NOS	2,32

Table 4.1: Case Study, mean values for each metric on the Refactoring Browser V 3.5.1.

LOC	5
MHNL	1
MNL	9
MSG	1
NI	1
NMAA	2
NOS	1

Table 4.2: Case Study, median values for each metric on the Refactoring Browser V 3.5.1.

modify.

4.3 Conclusion

Doing such evaluations helps the software engineer to get an idea of the quality of the code. To give some examples of how quality can be observed we can say that having small methods makes the code easier to read and to modify, having a low cohesion between different classes makes the code more flexible and too short method names can make the code more difficult to understand. Such an analysis is important in the software reengineering process, because it helps to distribute the available resources of a software company in the best way. The help of Moose with Momfe accelerates and simplifies this process.

LOC	110,21
MHNL	0,37
MNL	14,30
MSG	26,67
NI	20,30
NMAA	27,18
NOS	5,83

Table 4.3: Case Study, variance values for each metric on the Refactoring Browser V 3.5.1.

LOC	10,50
MHNL	0,61
MNL	3,78
MSG	5,16
NI	4,51
NMAA	5,21
NOS	2,41

Table 4.4: Case Study, standard deviation values for each metric on on the Refactoring Browser V 3.5.1.

	LOC	MHNL	MNL	MSG	NI	NMAA	NOS
LOC	1,00	-0,18	0,11	0,22	0,21	0,12	0,12
MHNL	-0,18	1,00	-0,05	-0,08	-0,11	-0,10	-0,12
MNL	0,11	-0,05	1,00	-0,02	-0,01	-0,00	-0,05
MSG	0,22	-0,08	-0,02	1,00	0,98	0,89	0,89
NI	0,21	-0,11	-0,01	0,98	1,00	0,93	0,91
NMAA	0,12	-0,10	-0,00	0,89	0,93	1,00	0,95
NOS	0,12	-0,12	-0,05	0,89	0,91	0,95	1,00

Table 4.5: Correlation Matrix of some method metrics.

	LOC	MHNL	MNL	MSG	NI	NMAA	NOS
LOC	X	O	O	O	O	O	O
MHNL	O	X	O	O	O	O	O
MNL	O	O	X	O	O	O	O
MSG	O	O	O	X	X	X	X
NI	O	O	O	X	X	X	X
NMAA	O	O	O	X	X	X	X
NOS	O	O	O	X	X	X	X

Table 4.6: Correlation Matrix. Values over 0.5 or under -0.5 correlate strongly. This relationship in the matrix is represented with an X

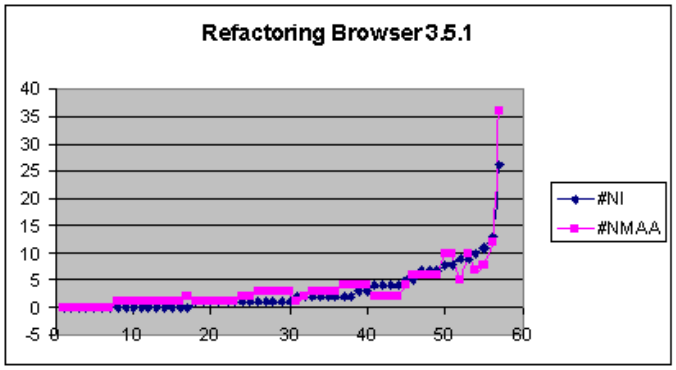

Figure 4.4: Correlation: This diagram shows the correlation between the NI and NMAA metric for the Refactoring Browser V 3.5.1. The on the x-ax one can see each methods in the model while on the y-ax one can see the related metric-value

Chapter 5

Conclusion

5.1 Momfe, conclusion

Momfe is a tool which provides some new interfaces to the Moose user. It take already existing functionalities and makes them more accessible. The user needs less know-how to work with metrics and this simplifies working with Moose. The exporting utility of Momfe makes it possible to use some external software to analyze the data in order to get another point of view on any software system. And this opens the way to new possibilities: combining Moose with other tools in order to extend the features of Moose.

Appendix A

Technical Details

A.1 Architecture

A.1.1 Introduction and Overview

All the functionalities of MOMFE were implemented in the application named MooseMetricsUIApp. This application contains following classes:

- *MooseModelMetricsUI*
- *MSEMetricDescriptionUI*
- *MSEMetricsManagerUI*
- *MSEMetricsTableConfiguratorUI*
- *MSEMetricsTableUI*
- *MSEMetricsTableExporter*
- *MSEModelInformation*

In the following subsections we will describe each class. We will present the structure of the class and its function in the context of the whole Momfe project.

A.1.2 The Class: MooseModelMetricsUI

The class which generates the Moose Model Metrics Viewer object is named Moose-ModelMetricsUI and its instance variables are:

- *tableInterface*
- *infoTableArray*
- *fillingSelector*

- *metricsInformator*
- *metricsTable*
- *isCompleteTableDisplayed*

The task of the Moose Model Metrics Viewer is to display a GUI with a table which visualizes the Number of:

- *Access Arguments*
- *Accesses*
- *Attributes*
- *Classes*
- *Entities*
- *Expression Arguments*
- *Formal Parameters*
- *Global Variables*
- *Implicit Variables*
- *Inheritance Definitions*
- *Invocations*
- *Methods*
- *Named Properties*
- *Objects*

What is the task of this Class? Basically with MooseModelMetricsUI Momfe asks the *Current Model* for the informations listed above and displays them in a table.

A.1.3 The Class: MSEMetricDescriptionUI

The class which generates the *Metrics Descriptor Object* is named MSEMetricDescriptionUI and its instance variables are:

- *metric*
- *description*
- *metricAttributes*
- *metricLanguages*
- *availableMetrics*

What is the task of this Class? MSEMetricDescriptionUI instantiates an object which displays information about the metric by directly asking the metric.

A.1.4 The Class: MSEMetricsManagerUI

The class which generates the *Metrics Manager Object* is named MSEMetrics-ManagerUI and its instance variables are:

- *availableMetrics*
- *languangeSelection*
- *selectedMetric*
- *filterString*
- *languageList*
- *entityList*
- *metricNamesList*

What is the task of this Class? The available metrics are directly taken from the MSEMetricManager-Object. The filter criteria simplify the view of the metric, which the user is looking for.

A.1.5 The Class: MSEMetricsTableConfiguratorUI

The class which generates the *Metrics Table Configurator Object* is named MSE-MetricsTableConfiguratorUI and its instance variables are:

- *entitiesToShow*
- *runOperators*
- *availableEntities*
- *metricsList*
- *chosenMetricsList*
- *metricsFilter*
- *availableMetrics*
- *languageList*

What is the task of this Class? This class instantiates a GUI which helps the user to chose the metric values, that will be displayed in the columns of the **EMT**, the *Entity Metric Table*.

A.1.6 The Class: MSEMetricsTableUI

The class which generates the *Metrics Table Object* is named MSEMetricsTableUI and its instance variables are:

- *tableInterface*
- *tableRows*
- *numberOfColumns*
- *columnLabelsArray*
- *aTableAdaptor*
- *tableMenu*
- *sortAscending*

What is the task of this Class? The main task of this class is to display a table with following data. For each entity of the chose entity kind the table displays a value of the chosen metric(s).

A.1.7 The Class: MSEMetricsTableExporter

The class which generates the *Excel Exporter Object* is named MSEMetricsTable-Exporter and it has no instance variables.

What is the task of this Class? This class simply creates an object which is able to generate a stream containing the values of the EMT and stores this stream into a file.

A.1.8 The Class: MSEModelInformation

This is the central class of the Momfe project, which implements most methods needed by the graphic user interfaces. It instantiates a singleton, in order to keep the all collected informations about the *Current Model*. And it sends these informations to any object belonging to MooseMetricsUIApp which will ask for it. The class which generates the *Moose Model Metrics Information Object* is named MSEModelInformation and its instance variables are:

- *currentModel*
- *availableMetrics*
- *metricsInTable*
- *entitiesInTable*
- *tableContents*

What is the task of this Class? Thus the instantiated object keeps Labels for the GUIs, fetches the ”list of metrics and the entities” available in the system, computes the metrics-values for each entity-kind, builds the table-informations which are visualized in the application, and keeps some information about the table itself. It also gives the number of access arguments, accesses, attributes, classes, entities, expression arguments, formal parameters, global variables, implicit variables, inheritance definitions, invocations, methods, named properties and objects for the *Moose Model Metrics Viewer*.

A.2 User Manual

This short manual should give an introduction how to run and use Momfe.

A.2.1 Before Running Momfe

Before using Momfe to display data on a Table or export them into any file the user should load some code into Moose and run the operators. After that the user can run Momfe.

A.2.2 Running Momfe

The command to start Momfe into the VisualWorks 3.0 environment is *MSEMetricsTableConfiguratorUI new open*. This command starts the *table Configurator* (see figure A.1).

This GUI helps the user to configure the desired table. After having configured the table the user can display it. To do that he just must click on the *Open Table* button.

As soon as the table appears (see figure A.2) the user can sort the values. This action is executed after double-clicking on the title of the desired column. The table-GUI can be reconfigured, exported to an external file or can be simply closed.

Note that in the configurator the user can ask for information about the acronym of a metric by double-clicking on the acronym.

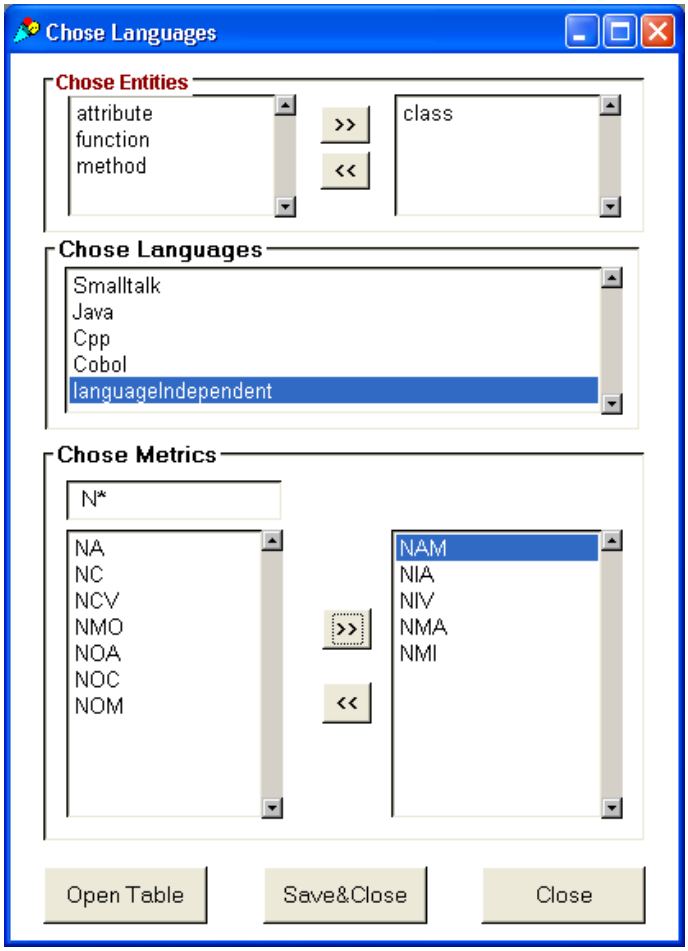

Figure A.1: Moose Metric Table Configurator: from here the user can chose the metric he wants to display

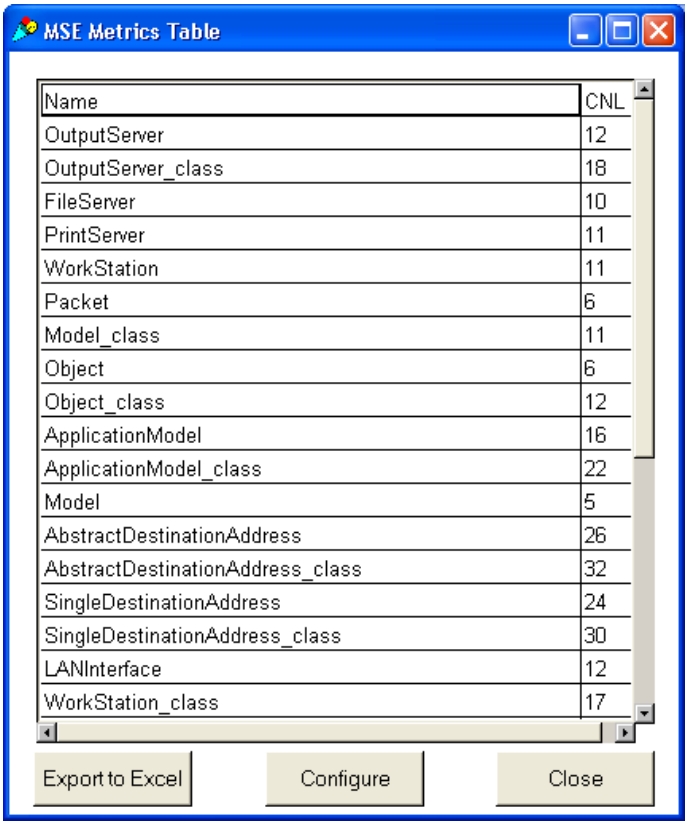

Figure A.2: The Entity Metric Table

Bibliography

[DD99] Stéphane Ducasse and Serge Demeyer, editors. *The FAMOOS Object-Oriented Reengineering Handbook*. University of Bern, October 1999. See http://www.iam.unibe.ch/~famoos/handbook.

[DDL99] Serge Demeyer, Stéphane Ducasse, and Michele Lanza. A hybrid reverse engineering platform combining metrics and program visualization. In Francoise Balmas, Mike Blaha, and Spencer Rugaber, editors, *Proceedings WCRE'99 (6th Working Conference on Reverse Engineering)*. IEEE, October 1999.

[DLT01] Stéphane Ducasse, Michele Lanza, and Sander Tichelaar. The moose reengineering environment. *Smalltalk Chronicles*, August 2001.

[KN01] Georges Golomingi Koni-N'sapu. A scenario based approach for refactoring duplicated code in object oriented systems. Diploma thesis, University of Bern, June 2001.