

# Exploring Size Metrics for Models

Panagiotis Tsakos  
s101571

DTU



Kongens Lyngby 2013  
IMM-MSc-2013-7

Technical University of Denmark  
Informatics and Mathematical Modelling  
Building 321, DK-2800 Kongens Lyngby, Denmark  
Phone +45 45253351, Fax +45 45882673  
[reception@imm.dtu.dk](mailto:reception@imm.dtu.dk)  
[www.imm.dtu.dk](http://www.imm.dtu.dk) IMM-PhD-2013-7

# Abstract

---

The goal of this thesis is to look into models from a different perspective, the metrics point of view. Since there are many model size metrics that can be exported from different models. We believe that this thesis is a good opportunity to create a tool that will try to measure and visualize them. We have the belief that with using it there might derive interesting finding. The contribution of our tool will help students and researchers to broader the academic views and give them a chance to explore the world of model size metrics!



# Preface

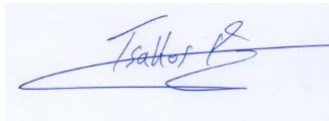
---

This thesis was prepared at the department of Informatics and Mathematical Modeling at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Computer Science and Engineering.

The thesis deals with measurement, visualization and exportation of model size metrics.

The thesis was developed under the supervision of Harald Störrle.

Lyngby, 01-January-2013

A handwritten signature in blue ink, appearing to read 'Tsakos P', with a large, stylized flourish extending to the right.

Panagiotis Tsakos  
s101571



# Acknowledgements

---

Firstly I would like to thank my family, and especially my mother Vasiliki Ntouni, my aunt Athina Kioussi and my uncle Ioannis Ntounis who made it possible for me to complete my studies at the Technical University of Denmark - Department of Informatics and Mathematical Modeling.

Finally, I would like to thank my supervisor Harald Störrle for his guidance and help ,throughout the development of this thesis.





# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Preface</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Background</b>	<b>1</b>
1.1 Problem statement . . . . .	1
1.2 Problem justification . . . . .	3
1.3 Related work . . . . .	4
1.3.1 Class metrics . . . . .	4
1.3.2 Use case metrics . . . . .	4
1.3.3 SDMetrics . . . . .	5
1.3.4 Sparx Enterprise Architect . . . . .	6
1.4 Background outcome . . . . .	6
1.5 Outline . . . . .	7
<b>2 Analysis</b>	<b>9</b>
2.1 Requirements . . . . .	9
2.1.1 Environment requirements . . . . .	10
2.1.2 Tool interaction requirements . . . . .	10
2.1.3 File import requirements . . . . .	11
2.1.4 File export requirements . . . . .	11
2.1.5 Model size metrics requirements . . . . .	12
2.1.6 Visualizations requirements . . . . .	13
2.1.7 User requirements . . . . .	13
2.2 User scenarios . . . . .	14
2.2.1 Student scenario . . . . .	14
2.2.2 Researcher scenario . . . . .	15

---

2.3	Use cases . . . . .	16
2.3.1	Use cases presentation . . . . .	16
2.3.2	Student use case diagram . . . . .	18
2.3.3	Researcher use case diagram . . . . .	19
2.4	Information model . . . . .	20
<b>3</b>	<b>Design of the plugin</b>	<b>23</b>
3.1	Design technologies . . . . .	24
3.1.1	Eclipse PDE . . . . .	24
3.1.2	Extensions . . . . .	24
3.1.3	SWT and layouts . . . . .	26
3.1.4	User interface . . . . .	26
3.1.5	Architecture . . . . .	31
<b>4</b>	<b>Implementation</b>	<b>35</b>
4.1	Technology . . . . .	35
4.1.1	Eclipse plug-in development environment . . . . .	36
4.1.2	SWT and JFace . . . . .	36
4.2	Plug-in development . . . . .	36
4.2.1	Plug-in project creation . . . . .	37
4.2.2	The “plugin.xml” file . . . . .	37
4.2.3	Views . . . . .	38
4.2.4	Pop Up Menu . . . . .	43
4.2.5	Perspective . . . . .	43
4.3	Libraries and components . . . . .	44
4.3.1	SDMetrics open core . . . . .	45
4.3.2	JFreeChar . . . . .	46
4.3.3	iText . . . . .	46
4.3.4	Window Builder Editor . . . . .	47
<b>5</b>	<b>Evaluation</b>	<b>49</b>
5.1	Testing stages . . . . .	49
5.1.1	Unit testing . . . . .	49
5.1.2	Functional testing . . . . .	51
5.1.3	Compatibility evaluation . . . . .	53
5.2	Requirements evaluation . . . . .	54
<b>6</b>	<b>Conclusion</b>	<b>57</b>
<b>A</b>	<b>Installation Manual</b>	<b>59</b>
A.1	Installation Methods . . . . .	59
A.1.1	Install as Project Repository . . . . .	59
A.1.2	Install Eclipse Plug-in manually . . . . .	67
A.1.3	Install from Online Repository . . . . .	68

**CONTENTS**

---

**ix**

**Bibliography**

**69**



# Background

---

This chapter presents the problem that our thesis addresses, justify it and afterwards we will follow with a review to the related work that focuses on the problem.

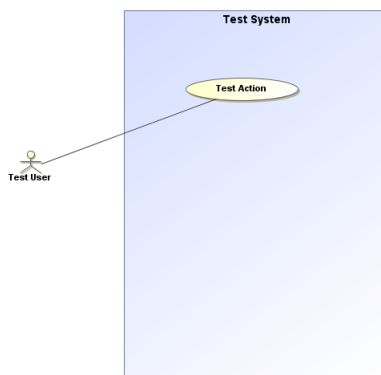
## 1.1 Problem statement

The problem we attempt to resolve by our thesis is the exportation and visual representation of model size metrics from different UML model files. Since there is no tool available that can easily perform model size metrics on different model files in order to allow students and researchers focusing on the field of system modeling to compare them and end up into useful results. We believe that the development of such a tool will be valuable.

There is an unlimited number of metrics that can be counted up by UML models but the ones that our tool is going to focus on are meta-class metrics, class metrics, activity metrics, component metrics, package metrics and usecase metrics. Which of these are metrics are useful? It is not completely clarified but with the development of this tool, we create the foundation on which different models can be measured and might end up in useful conclusions by our users.

In order to present the problem in a more simple way we will now give you a simple case study of how the models are being exported into an XMI file from a modeling tool and how it's metrics should be imported by our application in order to be further processed.

We have designed a very simple use case diagram using the MagicDraw modeling tool 1.1.



**Figure 1.1:** MagicDraw modeling tool - Use case diagram

From the diagram we can identify that we have an actor “Test User”, a system “Test System”, and action “Test Action” and an association between the actor and the action.

If we then export the above diagram into an XMI file we will have all these components into an XMI format that will look like this 1.2.

The XML language can successfully and efficiently represent all the meta-classes, elements and associations used in the UML use case diagram that we created. Using the model XMIs our tool should export the metrics out of it. In our case study we have the metrics of one use case metaclass, actor, system, action and association between the actor and the action.

These metrics could be then be available for our users to navigate through and visualize in different chart types and export them in different file types. It would be great if we had such a tool that with easiness and usability allowed us to perform all these actions.

```
<ndElement elementClass='Diagram'  
xmi:id='_16_5_4_74401a3_1359389739835_121913_463'  
name='Test Use Case' visibility='public'  
ownerOfDiagram='eee_1045467100313_135436_1'>  
<packagedElement xmi:type='uml:Actor'  
xmi:id='_16_5_4_74401a3_1359389745756_468594_485'  
name='Test User' visibility='public' />  
<packagedElement xmi:type='uml:Model'  
xmi:id='_16_5_4_74401a3_1359389751021_185171_504'  
name='Test System' visibility='public'>  
<packagedElement xmi:type='uml:UseCase'  
xmi:id='_16_5_4_74401a3_1359389772900_754195_518'  
name='Test Action' visibility='public' />  
</packagedElement>  
<packagedElement xmi:type='uml:Association'  
xmi:id='_16_5_4_74401a3_1359389785333_410390_535' visibility='public'>  
<memberEnd  
xmi:idref='_16_5_4_74401a3_1359389785333_781781_536' />  
<memberEnd  
xmi:idref='_16_5_4_74401a3_1359389785333_445125_537' />  
<navigableOwnedEnd  
xmi:idref='_16_5_4_74401a3_1359389785333_781781_536' />  
<navigableOwnedEnd  
xmi:idref='_16_5_4_74401a3_1359389785333_445125_537' /> |  
</packagedElement>
```

Figure 1.2: XMI example - Use case diagram

## 1.2 Problem justification

The reason we decided to go through the development of such a tool is to look into models from a different perspective, the perspective of their metrics. We know that the most commonly used methods for measuring the source code program size such as Source Lines of Code (SLOC) can not apply to UML models, therefore model size metrics is the obvious way that we should focus on. SLOC can be used in order to estimate the effort used to develop a program and programming productivity. Model size metrics can be used in order to measure the programming productivity and quality [DF11] when using model driven engineering (MDE) software development methodology. And according to “city inforum” the use of model metrics is even more important to numerous valuable applications in earlier stages of the development process like scheduling, cost estimation, quality assurance, and personnel task assignments. So a tool that both students and researchers can use in order to compare different model metrics might be essential. Easily exported metrics and visualizations can help to find more of what model size metrics can offer to us. Cross-screen comparisons will ease researchers to find patterns and associations between the model size metrics and end up into some very interesting outcomes.

## 1.3 Related work

There has been quite a lot of research in order to identify useful metrics. We believe it is important to look into some metrics that were found interesting and see what information they can offer.

### 1.3.1 Class metrics

Class metrics are mostly used in order to define the complexity of software. According to Shyam R. Chidamber, Object Oriented Design Metrics Suite[CK94], he identifies six metrics that can be measured by using the classes of a model.

These metrics are:

1. Weighted methods per class (WMC) - Metric regard to the complexity of a class method.
2. Depth of inheritance tree (DIT) - Metric that measures the length from the class to the root of the inheritance.
3. Number of children (NOC) - Represents the number of children and descendants of a certain class.
4. Response for a class (RFC) - Number of methods that can be invoked by an object of a given class.
5. Coupling between object classes (CBO) - Two classes are related when a method of a class uses and instance or method of another class.
6. Lack of cohesion in methods (LCOM) - Measures the number of sets of instance variables accessed by every pair of methods of a given class.

We believe that the above metrics are very effective in defining the complexity of software and we will try to support most of these metrics in our tool.

### 1.3.2 Use case metrics

Use Case metrics are in the other hand useful not only to define the complexity of a system but also to predict the work effort needed for this system to be



developed. On article by M. Marchesi[Mar98] he presents a set of use case metrics that are interesting to see.

These are:

1. NA - Number of Actors of the System
2. UC1 - Number of Use Cases in the System
3. UC2 - Number of communications among UC and Actors
4. UC3 - Number of communications among UC and Actors without redundancies
5. UC4 - Global complexity of the system

UC4 metric that represents the global complexity is calculated by the metric values of UC1, UC2 and UC3.

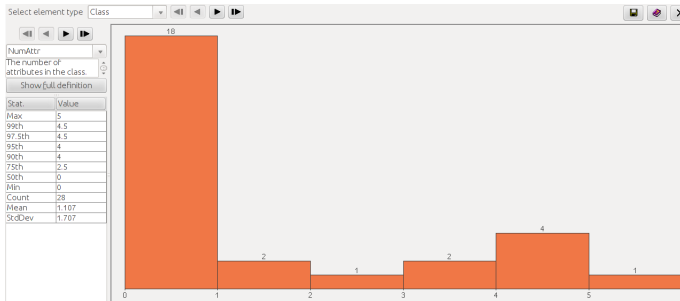
Our tool should also perform these metrics so that the users should be able to measure the complexity of a system.

### 1.3.3 SDMetrics

SDMetrics 1.3 [sdm]measuring tool is one of the few available measuring tools that can be used in order to analyze different types of UML diagrams. The tool can present the metrics information of a model file in the form of tables and histograms. Other features such as the Rule Checker can be used in order to find problems in the uml design. One more very interesting feature allows the user to define its own custom metrics and rules.

SDMetrics can measure metrics of class, activity, usecase and statemachine diagrams. It includes most of the metrics that were found to have some type of interest such as the class and usecase metrics we described previously, that define complexity and work effort. There is an open core version of SDMetrics that its code can be used for non commercial use, in order to implement and measure SDMetrics supported metrics in your own application.

We believe that SDMetrics is a very powerful tool but it does not provide many types of metrics visualizations such as scatterplots and bubble charts. As well it can not manipulate multiple model files simultaneously in order to visualize them and make it easier for the user to compare them visually.



**Figure 1.3:** An SDMetrics Histogram

### 1.3.4 Sparx Enterprise Architect

Sparx Enterprise Architect[spa] is a modeling tool but it is as well supports some use case metrics for predicting work effort and cost of a system modeled. The use metric features of Sparx might be interesting for a software development company since it can predict the cost of projects, but it might not be so useful for the users of our tool, since they want to explore model size metrics for academic reasons.

One drawback is that it does not support many different metrics except those use case metrics who are related with system complexity. It does not provide visualization of metrics in terms of charts and graphical comparison features. In the next figure we see Sparx making a prediction on estimated work effort and cost of system by using its use case metrics 1.4.

## 1.4 Background outcome

By working on this chapter, we reach into the outcome that model size metrics can truly help in earlier stages of development to understand the complexity of systems and predict the effort needed for developing them. Therefore model size metrics tools can also be very useful both in academic and commercial areas. The model size metric tools that are available are very useful but they do not support multiple types of chart visualizations of metrics and visual comparison of different models at the same time. For this reason we believe that the development of a tool that can help students and researchers to compare, visualize and export models is essential. We definitely believe that this thesis will be useful and will help students and researchers understand more of what model size metrics can offer.

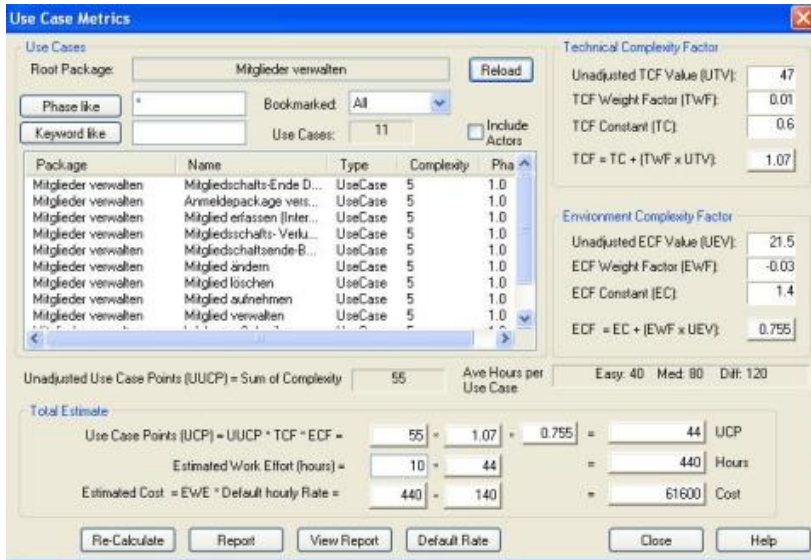


Figure 1.4: Sparx Enterprise Architect

## 1.5 Outline

This thesis is divided into five chapters and one appendix. The outline of the chapters and the appendix are presented below:

*Chapter 1 - "Background"* : In this chapter we describe the problem that our thesis wants to solve, we describe why this problem should be addressed and we also have a look at the related work that has been done.

*Chapter 2 - "Analysis"* : In the analysis chapter we will define the requirements we wish for our application and then make an analysis on them by using use case diagrams and other.

*Chapter 3 - "Design"* : In this chapter we will take a look at the implementation possibilities we have in order to make the correct choices for implementing our tool. We will also try to design the GUI of our tool.

*Chapter 4 - "Implementation"* : In this chapter we will try to explain how the implementation phase was done in more advanced and technical matters.

*Chapter 5 - "Evaluation"* : In this chapter we will try to evaluate our work so far. Give some testing examples and finally perform a requirements evaluation.

*Chapter 6 - "Conclusion"* : We will conclude our thesis in this chapter.

*Appendix A - "Installation Manual"* : Installation manual for our plug-in.

## CHAPTER 2

# Analysis

---

In this chapter we will look and analyze the problems stated in the previous chapter.

We are starting by describing the requirements we define for the tool that we wish to develop. The users of the plug-in are going to be mentioned and use cases will be presented.

### 2.1 Requirements

Let us now look into the requirements that we set for the development of our tool. The tool should be able to comply with the majority of the requirements and we are going to evaluate that in the final chapter of our thesis “cite evaluation”. We will try to be as more analytic as possible to help the reader understand what, how and for whom we want to do something.

### 2.1.1 Environment requirements

The requirements that we start with are for the environment that we want our tool to be based. We decided that the environment that we want our tool to be based on is Eclipse. Therefore our tool will actually be an eclipse plug-in that performs model size metrics on different model files. The reason we decided to go on with that possibility is because Eclipse allows to use its own extensions and components to create a usable and flexible graphical user interface for our tool. The Eclipse allows developers to create plug-ins by using Eclipse Plug-in Development Environment (PDE) which is based on Java and uses Standard Widget Toolkit (SWT) that ensure compatibility in all types of Operating Systems (OS)[EC08].

### 2.1.2 Tool interaction requirements

The interaction with the tool will be developed using eclipse PDE extensions. These extensions should ease the user to navigate through the tool environment in order to import and maintain model files, pass their metrics through different components, visualize them and export them. Such interactions features should be:

- Project explorer - With which the user will be able to import and maintain his model files for the cause of performing model size metrics on them.
- Measurements and visualizations viewer - A simple viewer from which the user can see which measurements, when and on which files have they been performed. Which visualization have been created and under which measurements they are based on. He should also be able to navigate through the different measurements and navigations and delete them.
- View components flexibility - The components viewing something in the interface of the tool should be able to be closed, be created and be moved around the environment of the plug-in. This is essential because if we allow the view components to be moved around in the environment, will help the user to adjust the environment in his own needs and view different multiple metric visualizations on his screen and perform cross-screen comparisons.
- Drag and drop - Files should be able to be dragged and dropped into the project explorer than just getting imported.

### 2.1.3 File import requirements

Let us now have a look to the model files that our plug-in should be able to import and perform model size metrics on. Most modeling tools such as Magic-Draw are able to export the models that were designed by using them in XML language. More specifically the exportations are made using XML Metadata Interchange (XMI) which is a standard for exchanging metadata information via XML language. XMIs are ideal for exporting UML models in a serialized way. So the requirements that we set for our tool is to support importation of model files in the XMI format for UML. The versions of XMI and UML we want to support are

- XMI: 1.0,1.1,1.2,2.0,2.1
- UML: 1.x,2.x

The actual difference between UML v1.0 and v2.0 is that the older one supports less metaclass types, so it is obvious that we can measure less model size metrics from an XMI file using UML v1.x than one which uses v2.x .

One more important issue concerning import requirements that we should set is, that since the meta-class metrics that can be measured from a model file are so many and perhaps our user does not want to get metrics for all of them. A filter file feature should exist which should be written in Comma Separated Value (CSV) format. This filter file should state all the user desired meta-class metrics and the tool should then parse it and only measure the desired ones.

It is also important and we state it as a requirement, that our tool should be able to import multiple model files and perform model size metrics on them for comparison reasons.

### 2.1.4 File export requirements

Now that we defined the import requirements, we can look into the export ones. Since our tool will be handling metrics and visualizations, it is very important that the users could export these in order to save them and reuse them whenever they want it to. Concerning the model size metrics, we can define as requirement that the tool should support their exportation in two different types of files:

1. CSV files - The metrics that would be exported in CSV format might be

very useful since they can be easily reused from spreadsheet programs such as Microsoft Office Excel for further processing.

2. PDF files - It would also be good if the plug-in could export the metrics of different models in a PDF file, which suits perfectly when the user want to export metrics for reporting reasons.

On the other hand the users of our tool should also be able to export different visualizations created by it. Of course the most suitable way to export visualization is in image files. So we define as requirement that the visualization created by our tool should be able to get exported into two types of image files:

1. Images based on pixel format: These are the most commonly used image format that is used by most computer users and they should be supported. These image file types are JPEG, PNG and many more.
2. Images based on vector format: These images are more suitable for those who need to resize the images and not lose any quality.

There should also be a possibility for the user to print the visualization from the GUI.

### 2.1.5 Model size metrics requirements

Model size metrics requirements are difficult to define by ourselves which are the most important ones, so we will try to adapt the requirements that other model size metrics tools define and as well allow the user to define meta-class metrics by themselves. We have looked in what other MSM tools support in the related work section(cite Related work). From what it was mentioned in the articles we studied, we can define as required metrics:

- Class metrics - Which can help our users to predict the software complexity.
- Use case metrics - Which can help our users predict the system complexity and the work effort required for system development.
- Meta-class metrics - Which can also help our users to measure productivity of models designed. For these type of metrics as we mentioned before, we will let the users to specify by themselves which should be measured.



### 2.1.6 Visualizations requirements

By visualizations requirements we mean what types of visualizations our tool will be able to draw for to the user. The visualization must also be able to present model size metrics measured from multiple model files. The visualizations that we define are:

1. Table views - A table which will include all the metrics measured from a model file. Table view is very useful because the users can see all the metrics derived from each model file and compare them with other table views or visualizations.
2. Histograms - A histogram will give the ability to the user to easier compare the size of different metrics, the size of each metric from one model file with the same metric from another model file.
3. Scatterplots - This type of chart will allow the user to combine two different metrics or meta-classes and compare them with others on a scatterplot drawing.
4. Bubblecharts - This is a three dimension chart that can help the users to compare three different metrics or meta-class types. It is that chart that you should use when you want to perform more complex graphical comparisons.

Whenever we measure metrics for multiple model files, there should be a way that the user can even visualize the addition of those metrics than visualizing multiple model files in a single visualization.

### 2.1.7 User requirements

Let us now look in the requirements that our users should have in order to use our plug-in. We can divide the users of our application into two different user categories.

1. Students - The students and most preferably master students who are interested in conducting metrics for the course (02341 -Model Based Software). Those student may want to explore model size metrics for educational related matters. The tool will help them to understand the basic uses of model size metrics and perform some measurement themselves.

2. Researcher - The researchers who have want to look into different model size metrics, do simple and more complex comparisons and end up with some results that have scientific interest.

The general requirements that we can define for both of these categories are the following:

- All user should have some basic knowledge in system modeling and more specifically UML modeling in order to understand the measurements and visualizations that the plug-in can perform.
- The plug-in interface is written in English so an average knowledge of English language is a requirement.
- Average knowledge of the Eclipse IDE is required since the users will use components of Eclipse to perform Model Size Metrics.

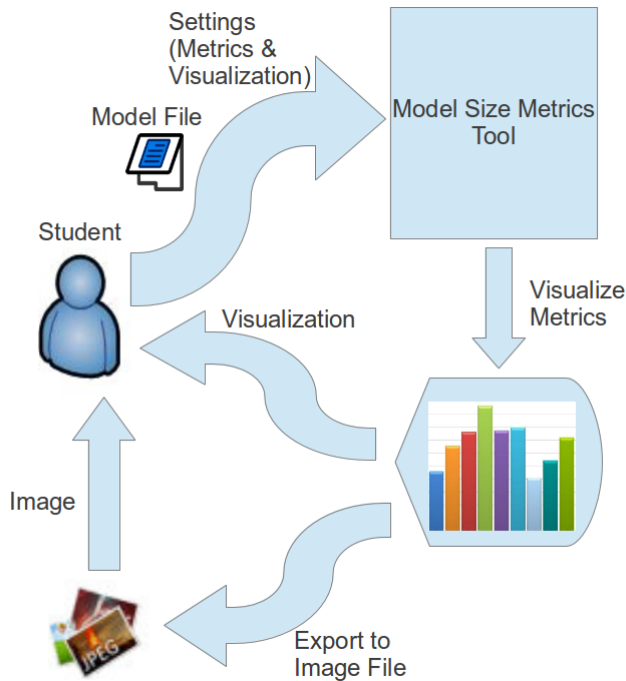
## 2.2 User scenarios

As we defined previously there will be two user categories for our tool the students and researchers. In this section we will try to demonstrate in two scenarios how each one of these user categories will interact with the tool.

### 2.2.1 Student scenario

Let us create a simple scenario for students, since we expect them to have less knowledge in the field of system modeling than the researchers. We expect students in our scenario to only perform simple actions with our tool. In the figure 2.1 we made a drawing that tries to demonstrate exactly that scenario.

As we see in the figure 2.1 there is a student which interacts with the tool and provides it with a single model file, metric settings, visualization settings and as well he asks to export it in a pixel based image file. By metric settings we mean what type of metrics he wants to be measured and be provided to him. By visualization setting we mean the type of visualization in this scenario it is a histogram. In the next phase according to the flow of the drawing the tool is generating the requested metrics and shows their visualization on the display. Then a pixel based image file with the visualization is exported and it is delivered to the student.

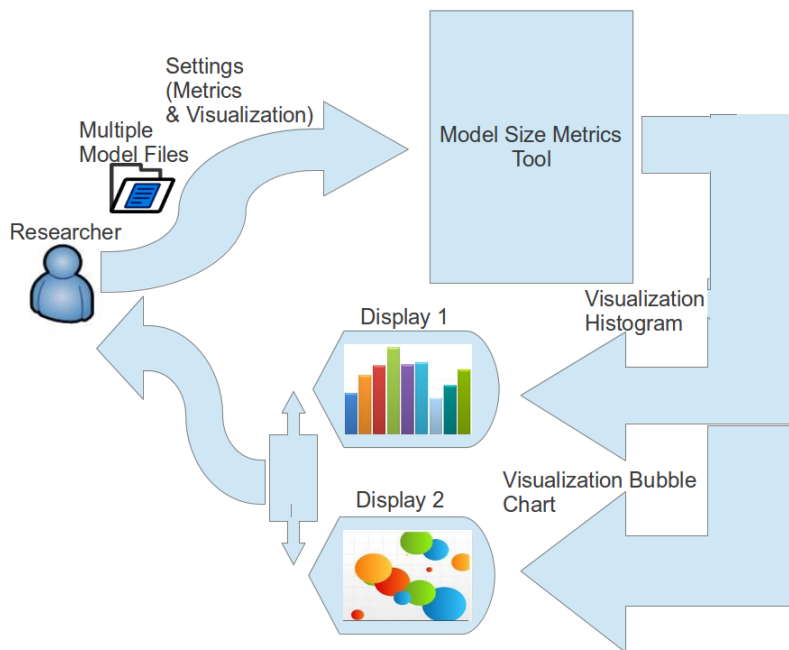


**Figure 2.1:** Student scenario

### 2.2.2 Researcher scenario

For the researchers user category we are going to create and draw a more advanced scenario. The scenario will present two types of visualizations a histogram and a bubble chart. We decided to use bubble chart in this scenario since it is the most complex chart type that our tool supports as it is a three dimensions chart. Each bubble location in the chart depends on two different metrics (x and y) and its size - ratio ( $\pi$ ) is defined by one more metric. The figure 2.2 tries to demonstrate that more advanced scenario.

As we can see from the figure 2.2 in this scenario we have a researcher that interacts with model size metrics tool. He provides the tool with multiple model files and his metric and visualization settings in order to demonstrate all those models in a single bubble chart in one display. He also creates a histogram which appears in the second display. He can compare the visualizations of both displays by using Eclipse cross-workbench feature.



**Figure 2.2:** Researcher scenario

## 2.3 Use cases

In this section we will write down all the use cases that we want our tool to implement and afterwards we will try to provide in use case diagrams for all the actions that each one of the user categories can perform on our system. The actions will be presented as use cases, the users as actors and our model size metrics tool as the system.

### 2.3.1 Use cases presentation

- **UC-ID: Import Model**

1. Description: Action that allows single or multiple model file imports into the system
2. Scenario: The user has some model files that he wishes to find their

metrics. With this action he can import these files into the system

- **UC-ID: Get Metrics**

1. Description: Action that allows to get the metrics from a single or multiple model files
2. Scenario: The user has already imported his model files into the system. He now wishes to get their metrics. He can just use this action to do so

This use case requires Import Model as precondition

- **UC-ID: Show Visualization**

1. Description: Action that the user takes when he wants to get the metrics as a visualization files
2. Scenario: The user has already got the metrics out of his model files and he now wishes to visualize them in a histogram. He takes this action in order to do so

This use case requires Import Model and Get Metrics as preconditions

- **UC-ID: Export Metrics**

1. Description: Action that the user takes when he wishes the metrics to be exported in some type of the supported format
2. Scenario: After user have got the metrics from a model file by using this action. He can export them into CSV or PDF format

This use case requires Import Model and Get Metrics as preconditions

- **UC-ID: Export Images**

1. Description: Action that the user takes when he wishes to export a visualization as an image
2. Scenario: After user have got a visualization from the tool he can then use this action in order to export it in pixel or vector based image formats

This use case requires Import Model, Get Metrics and Show Visualization as preconditions

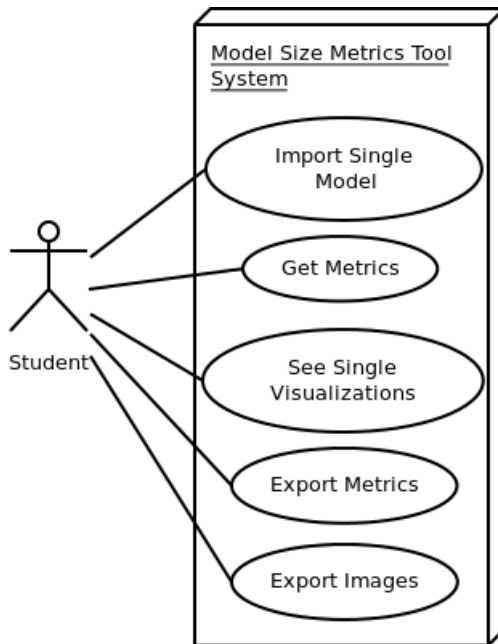
- **UC-ID: Compare Visualizations**

1. Description: The user might create multiple visualization and he can compare them using this action
2. Scenario: The user might want to have multiple model file metrics in a single visualization in order to compare them. Or might have multiple visualizations that he compare cross-platform

This use case requires Import Model, Get Metrics and Show Visualization as preconditions

### 2.3.2 Student use case diagram

The use case 2.3 diagram tries to present all the actions that a student might perform on our tool always according to the requirements that we defined and the user scenarios that we described previously.



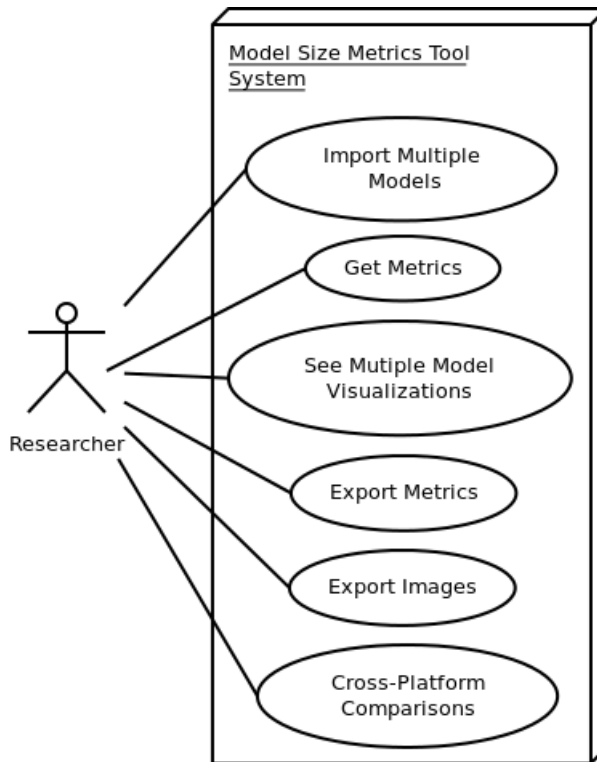
**Figure 2.3:** Student use case diagram

By looking into the use case diagram we identify some actions that the student user can do with our system. These actions derive from the requirements and

do not limit the student to perform even more advanced actions that might not be stated in the usecase diagram. The reason we separated the actions that can be performed into two different user categories is just because we try to demonstrate which actions are more appropriate for a student user, which more often will not need to perform complex actions with our tool.

### 2.3.3 Researcher use case diagram

The figure 2.4 tries to demonstrate the actions that are more appropriate for a researcher to perform using our tool. Which they should definitely be more complex.



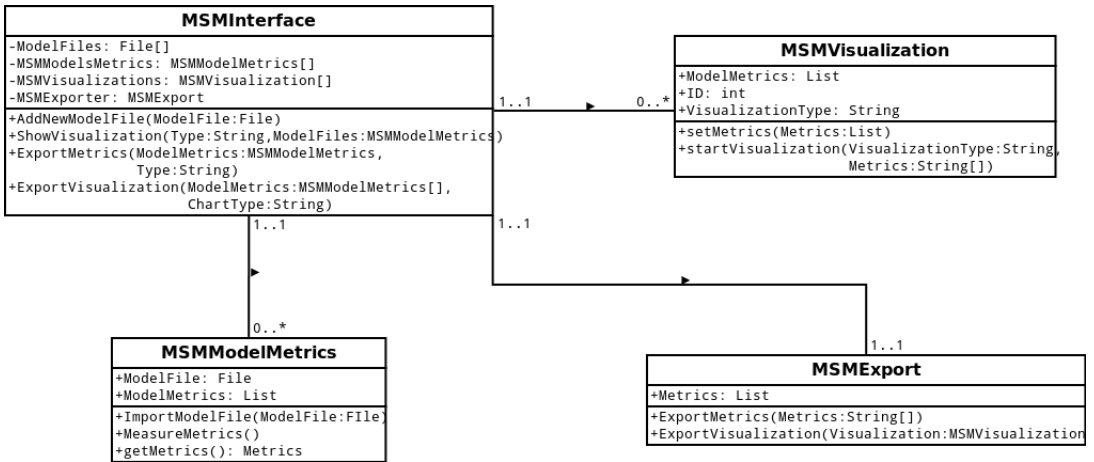
**Figure 2.4:** Researcher use case diagram

The researcher user will mostly try to compare multiple model files and perform visualizations that represent multiple metrics. We expect the researcher to investigate different metrics that have the potential to bring up useful conclusions.

And that he will be using a lot the cross-workbench comparison feature for his research.

## 2.4 Information model

The goal of this section is to sum all the requirements and use cases that we described in this chapter and provide an analysis level class diagram 2.5 that includes the most major ones.



**Figure 2.5:** Analysis level class diagram

As you can see from the class diagram we separated our tool into four different classes that will implement our requirements.

- **MSMInterface** - This should be the main class of our tool. It should implement the environment of the tool and have as attributes all the objects created by the rest three classes. It implements some general methods that perform actions such as “Import New Model File”, “Show Visualization” and “Export Metrics or Visualizations”.
- **MSMModelMetrics** - Is a class that we add each one of the model files loaded to system and then use its method “MeasureMetrics” to perform and store model size metrics as its attributes.
- **MSMVisualization** - Is the class responsible for visualization metrics into table views or charts and then show them to the users. It has as methods



---

“Set Metrics” which actually gets the metrics that we wish to visualize and “Start Visualization” with a visualization types as parameter to specify what type of visualization we want.

- MSMEExport - Is a class that care of the exportation both for metrics and visualization. It gets as attributes a list of metrics and it can then exports it in CSV or PDF format. It also has a method names “ExportVisualization” that takes as parameter and MSMVisualization object and export its visualization as different images.



## CHAPTER 3

# Design of the plugin

---

In this chapter we will go through different design technologies that we looked into in order to design our plug-in. We will explain and justify our choices. We will go through eclipse Plug-in Development Environment (PDE) tools that helped us to create our application. There will be presentations concerning the design of the plug-ins Graphical User Interface (GUI). The sketches will be shown that were used as GUI design drafts in the early stages of the development of this thesis.

Through this chapter we will try to satisfy all the requirements that were defined in the analysis chapter and as well provide usability to our users in the most efficient way.

In the end of this chapter we will demonstrate the data structures that we created for metric storing and retrieving. And as well explain in simple examples the algorithms and procedures that should be implemented in order to achieve the desired outcome.

## 3.1 Design technologies

As design technologies we mean all the technologies, tools, components available that can help us design our tool. In order to start we must begin first by looking at the environment that we will be using to design our plug-in and this is PDE. PDE is the environment of Eclipse that allows developers to create Rich Client Platforms (RCP) and Plug-ins for eclipse.

### 3.1.1 Eclipse PDE

For the development of our tool we first had to see through the different design options we had and then justify why we decided to design an eclipse plug-in. The actual difference between plug-ins and RCP applications is that RCP applications are stand-alone applications implementing the same tools and components that plug-ins use. This means that the implementation of RCP application is pretty much the same with the plug-in implementation, with the difference that RCP actually runs in stand alone basis in contrast with plug-ins that runs as embedded extensions of the eclipse environment. The reasons we decided to design a plug-in that will perform model size metrics are:

- Because it is better for our case to use some of the already provided eclipse functional and reliable components such as the package manager than creating our own from scratch.
- And as well because we would like to use Eclipse workbench to stack, move around, minimize and close the different views that we will design.

Eclipse PDE provides different types of components that we can extend in order to reach our design goal[EC08]. These components are referred in by eclipse as extensions. There are so many different types of extensions that we can utilize for designing our tool. We of course not going to explain all of them but we will have a look into the most important ones.

### 3.1.2 Extensions

In this section we will see the mostly used plug-in extensions and more specifically focus on the ones that we used in order to design our plug-in.

The most important and most commonly used extension is eclipse PDE is the view.

### **3.1.2.1 Views**

The view is actually an empty window inside Eclipse IDE which we can fill up with widgets needed for performing different actions. The view is also very useful for our tool development, since it will allow us to fill them up with visualizations and then be free to move them, close and stack them everywhere around the IDE and this way perform cross-workbench comparisons.

### **3.1.2.2 Pop up menu**

The pop up menu is an extension that places an extra menu entry when a right click is occurred on a file that was imported while using the eclipse's package manager. We will include that extension in the design of our tool since a file manipulation view such as the package manager, will allow us to import and perform measurements on different model files. A pop up menu can filter and only provide actions to the files specified by the extension.

### **3.1.2.3 Perspective**

Perspectives are extensions that help us pre-define the initial positions that each view will be placed in the API. The perspective extension adds a perspective entry in window menu of eclipse from where we can open it. It is very functional because without we should have opened all the views required for our tool to function manually. We will definitely use a perspective extension for our plug-in.

### **3.1.2.4 Menu**

The menu extension can add menu entries on Eclipse's main menu or tool bar. We had used menu extension to add a tool bar button on eclipse's API which whenever clicked it was opening our tool's perspective. Since that was not very functional we decided to drop it and not use it as part of our design. Since it does not make any significant contribution and eclipse IDE by default provides a very simple and functional way for opening perspectives from its menu.

### 3.1.3 SWT and layouts

Since the Eclipse plug-ins graphical widgets that by using them we will allow our users to perform different actions on our tool are constrained to function only under Standard Widget Toolkit (SWT) technology. We will give a short explanation about what SWT widgets actually are and how we are going to include them in our design. And then we will discuss about the available layouts a view extension can use, in order to place the different widgets on its body.

#### 3.1.3.1 SWT

SWT is a graphical widget toolkit that supports plenty of widgets which implement many and very usable functionalities. Such widgets are buttons, combo boxes, text boxes, tables, trees, labels and generally all these tools that a user can get information from or by interacting with them perform actions with the plug-in. Another big advantage of SWT except the usability that it provides, is the native look that guarantees up to certain point, that the look of the design will be pretty much the same when our plug-in will be running in different operating systems.

#### 3.1.3.2 Layouts

In order the for the widgets to be placed correctly in a view and more precisely in a SWT composite. We must first assign a layout to that composite that will position its children “other SWT Widgets” inside it. There are many different types of layouts available and the most standard ones are the FillLayout, the RowLayout, the GridLayout and the FormLayout. Since all these layouts create constrains in the freely and with no limitations positioning and sizing of the widgets. We decided to use as a layout for our plug-in the so called absolute or null layout that allows to freely place and size each widget simply by defining its bound properties. Except the visualization views because of JFreeChart library constrains.

### 3.1.4 User interface

The user interface of our plug in is going to ensure usability for our users and we are going to explain the process of designing it in this section. We will first start by presenting the design that we have made in the beginning stages of

development. We will argue about it and explain why we rejected it. Finally we will show sketches of the final design template as well with explanations regarding its functionality.

3.1.4.1 Early stages of design

The first tries for designing the user interface were done on a image drawing software. The figure 3.1 shows the drawing of how user interface was suppose to look like inside eclipse’s IDE.

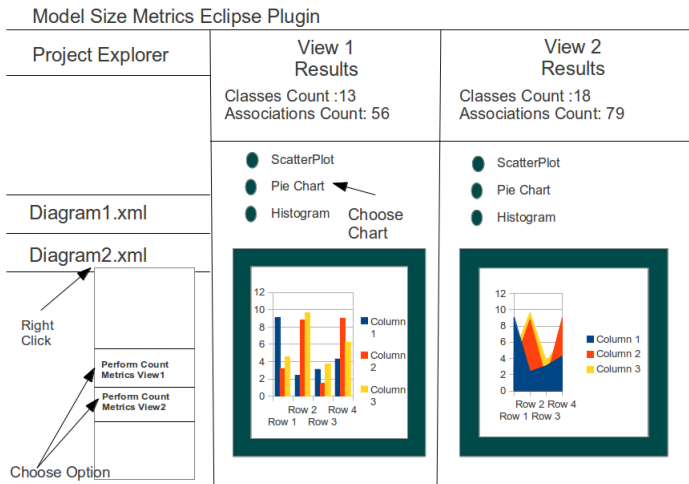


Figure 3.1: User interface drawing

As we can see from the figure on the left side of the drawing the project explorer of eclipse is shown. The project explorer was going to act as a file browser for our plugin, where the users could import their model files that they wanted to perform count metrics on. If the file the user have chosen is not appropriate an error message appears stating the error that the file was invalid. The plugin would starts to perform the count metrics by right clicking on the model file and then a pop-up menu would appear with the option to “Perform count metrics” on different view tabs. The view tabs appear on the right side of the drawing

where the count metrics results for each file is shown and there are also options of visualizations we want to perform on those results.

The previous described design was rejected for the following reasons:

- It does not provide multiple model file measurements and visualizations.
- It does not give file related information to the user that might be useful such as the time last modified, the date last modified, the model file XMI and UML version.
- It does not provide an easy way to see and navigate through the metric measurements and visualizations created.
- It does not allow us to have unlimited amount of measurements and visualizations since the total amount of views that can be created is defined.

We will now in the following sections provide you with the sketches that we drawn for the final version of our tool.

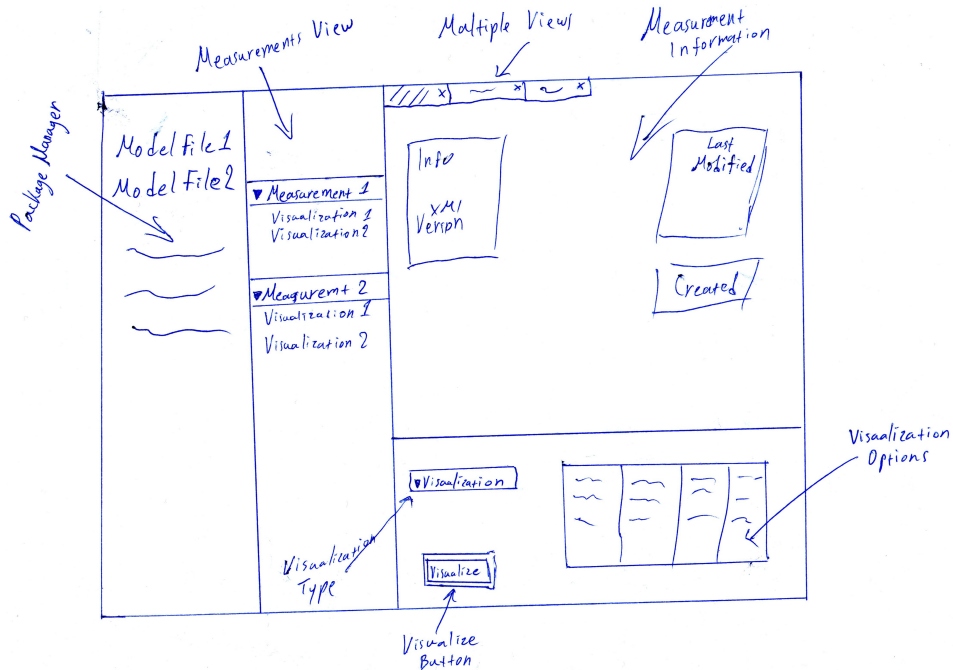
#### **3.1.4.2 Perspective and measurements user interface design**

As we can see from the sketch 3.2 the final plug-in perspective design will look like this. On the left size of the perspective there will be the package manager from where the user can import all the model files that he would like to perform model size metrics and visualize them. The actual interaction between the user and package manager except importing, will be the ability to select a single or multiple model files using (CTRL) key from the keyboard and then by right clicking on them from the pop-up menu to choose the measure action. The measure action will parse and measure the metrics from the single or multiple files and then register it as a measurement on the measurements view next to it on the right side.

The measurements view will show to the user all the measurements that were done and also work as a navigation bar. Whenever a measurement in the measurement view is selected the plug-in should activate it and open it in the right side of the perspective where measurement's information and visualization views can stack.

In this sketch we present the appearance of the measurement information view. The measurement information view on the top holds model file related information such as name, path, last time, last day modified, XMI and UML versions.





**Figure 3.2:** Perspective and measurements sketch

This way we inform and the user know if the file or files measured are the ones he actually wants. At the bottom of this view the user can select visualization types that he wants his data to be visualized.

Such types can be as we described at the analysis part:

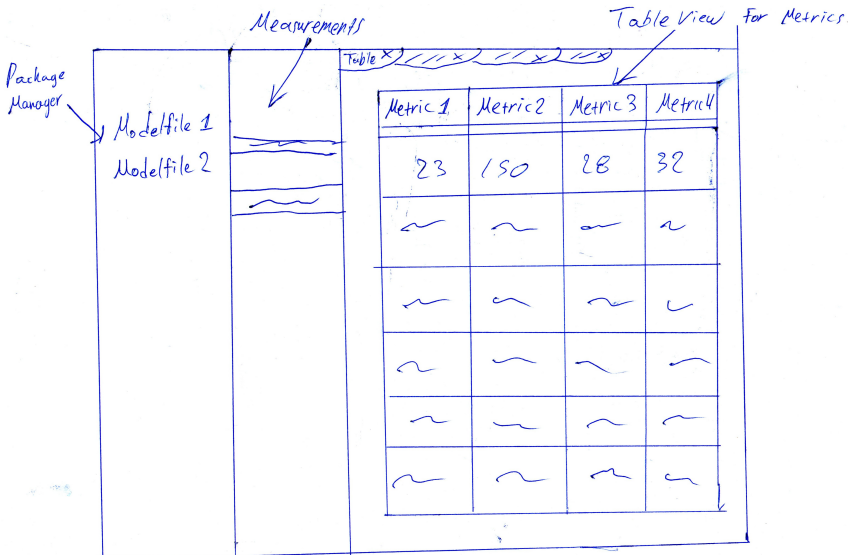
- Table View - Including the metrics
- Histogram
- Scatter Plot
- BubbleChart

There is also a table that the user can put more specific settings of the visualization he wants. Such as the color of a dataset series, sorting options, specify metrics to be visualized in a scatter plot and other.

Finally there is a button named visualize which when clicked it creates the visualization defined by the settings and present it in a different view. Whenever

a new visualization is made it adds up a child under the measurement object at the navigation located at the measurements view. This way the user can also navigate through the open visualizations. Whenever a measurement is closed its children visualizations will also get closed.

### 3.1.4.3 Metrics table user interface design



**Figure 3.3:** Metrics table sketch

The metrics table view can be seen from the sketch 3.3 we provided. The table has as goal to present model size metrics such as metaclass, class, activity, usecase, component, package and other diagram metrics to the user. The table itself is a type of visualization show it can be shown just like all the rest types of visualizations. Each metric name can be identified by the name of each column and each row represents objects of a specific metric type. The metrics table view will also appear in the navigation menu and it can be closed or moved around the IDE.

### 3.1.4.4 Visualizations user interface design

In the sketch 3.4 we try to show the design of a scenario where having two chart visualization one Histogram and one Scatterplot created. The chart visualiza-

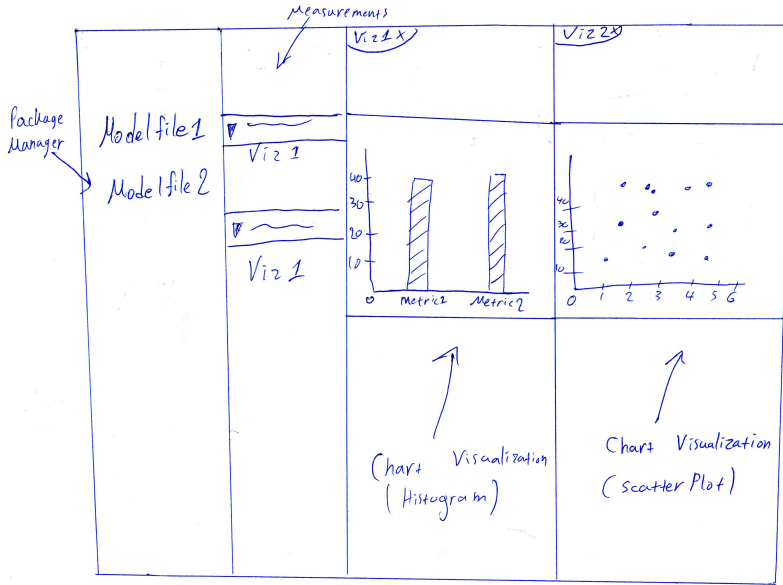


Figure 3.4: Visualization sketch

tions will fit in another type of view that we call visualization view. Such views when created can be moved everywhere around the IDE and in this scenario they are placed side to side for cross-workbench comparison. Like tables they will also be added at the navigation toolbar of the measurements view.

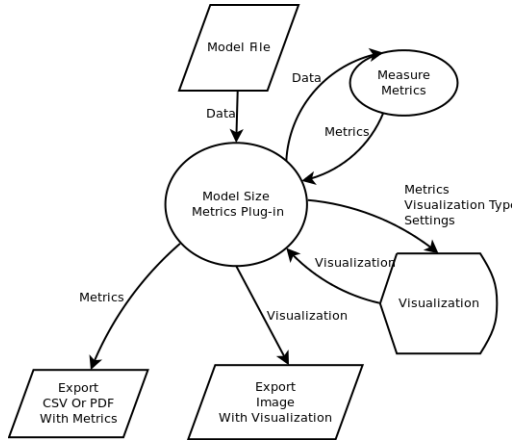
### 3.1.5 Architecture

In this section we will look into different design diagrams that will try to illustrate the architecture of our plug-in.

#### 3.1.5.1 Data flow diagram

A data flow diagram will try to explain to you the flow of the data that will be manipulated by the plug-in we are designing.

The figure 3.5 tries to give us a general idea of how the main plugin handles the data flow for importing and exporting data. In the beginning we start with an input which is a model file. The model file data are then flown to the MSM



**Figure 3.5:** Data flow diagram

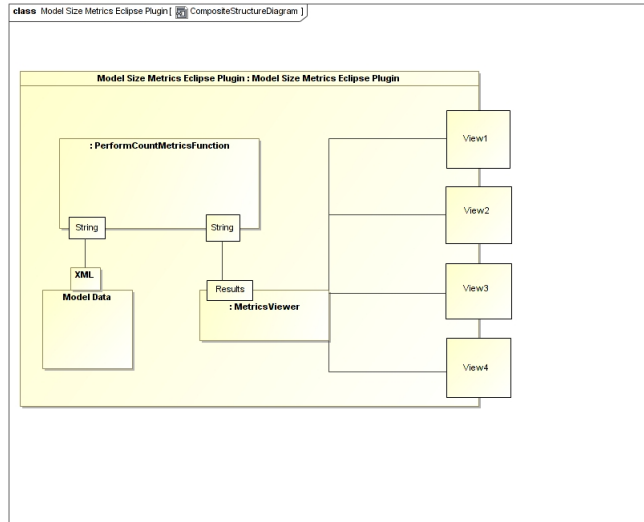
plug-in and the plug-in passes this data to the metrics measure functions which is then returns to MSM plug-in the metrics. Then the metrics can even get exported in a CSV or PDF file or sent together with settings for visualization to the display. Then the visualization is sent back to the plug-in which can then export it as an image file. For sure the exportation of data feature should be present in the plug-in interface implementation.

### 3.1.5.2 Composite structure diagram

The next diagram 3.6 that we designed tries to demonstrate the interaction between the processes that are responsible for visualizing metrics in different views at the runtime. We can identify parts that play a role in the visualization such as the model file, the perform count metrics process and the metrics viewer process which are actually responsible for visualizing the metrics. And finally we have ports named in the diagram as views from which we are actually presenting the visualizations to the user.

### 3.1.5.3 Data structures

It is important in the design phase to try to demonstrate how the data structures will be. We definitely are in need of good data structures since we will have to



**Figure 3.6:** Composite-Structure diagram

manipulate a lot of metrics data. When the data will be exported should be stored in dynamic and easily accessible way. For this way we decided to store the metrics that the plug-in will manipulate internally in a HashTable data structure.

The reason we decided that is because hash tables are good to handle large amounts of data and can drastically minimize the searching of an entry. Also model size metrics fit perfectly in a hash table since we can pass the metric name as key and the size as value.

Hash tables will also allow us to easier sort data according to our needs, simply just be transferring them to list and then specify comparators on the keys or values that will do the job for us.

#### 3.1.5.4 Deployment

The set up of the plugin is suppose to be simple and follow every eclipse plugin deployment standards. The plugin is going to delivered in different ways.

- As a single deployable jar file that can be loaded on the eclipse platform and add the plug-in on eclipse.

- As a zip file that contains all the project and can be imported in eclipse as plug-in project. Therefore it could be run as an eclipse application that includes the plug-in.

Together with the final report delivery an appendix will be included giving the user all the instructions of installation and use.

# Implementation

---

The plugin that we have developed for our thesis passed through many implementation steps. We will describe these steps in order to give better and clearer understanding of the technologies we used, the plug-in development, code structure and the components and libraries used to get to the final result. This chapter focuses to make readers with advanced background in software engineering and also other students that would like further develop it to fully understand the development process.

## 4.1 Technology

The technologies we used for implementing our tool which more or less were described in the design chapter of this thesis. In this chapter it will be described from a more advanced software engineering aspect in this section.

Our model size metrics tool was implemented using Eclipse Plug-in Development Environment (PDE), because it would contribute more to our goal by using eclipse's usable Integrated Development Environment (IDE) user interface (UI) than creating our own from scratch. As programming language that is used is Java.

### 4.1.1 Eclipse plug-in development environment

Eclipse PDE will allow us to build our tool with all the advantages Java and Eclipse IDE provides. Eclipse is an extensible IDE, everybody is able to extend and customize its platform by creating a plug-in. Eclipse architecture for plugin development is based in three components[EC08].

1. Eclipse Platform
2. Java Development Toolkit (JDT)
3. Plug-in Development Environment (PDE)

### 4.1.2 SWT and JFace

Eclipse PDE extends Eclipse abilities using “extensions”. These extensions use a graphical widget toolkit named Standard Windows Toolkit (SWT). SWT is very important component for plug-in development since we can create the graphics that are going to be attached on the Eclipse platform by using it. SWT was designed by eclipse because the existing Abstract Windows Toolkit (AWT) and Swing did not perform well enough and did not provide native application look and feel. SWT solved this problem with success and we can now enjoy high performance on graphics on all of our plug-in extensions.

JFace is a high level framework that is functioning on top of SWT to make it easy to use. It provides helpful classes for handling and implementing SWT UI features that might be difficult to do without it[EC08].

## 4.2 Plug-in development

This is the section explaining the steps, procedures, plug-in components and programming techniques that were used for the development of our plug-in.

In order to start we should present the initial steps needed to create our plug-in in Eclipse PDE.



### 4.2.1 Plug-in project creation

To create our plug-in project we first start by creating a new plug-in project in Eclipse. Eclipse wizard creates all the basic plug-in project folders and files which are:

- The “plugin.xml” file: Defines the extension aspects of the plug-in.
- The “META-INF” folder: Which contains the MANIFEST.MF file.
- The “MANIFEST.MF” file: Defines the runtime aspects of the plug-in.
- The Activator class: Represents the plug-in from a programmatic standpoint.
- The “build.properties” file: Includes building configurations and runtime information.
- The “src” folder: Where we should create our packages and implement our classes.

The “plugin.xml” file is a very important file for the plug-in development in Eclipse and we will have a closer look into it.

### 4.2.2 The “plugin.xml” file

The “plugin.xml” file is a file using XML format to specify plug-in extensions. It can be edited as a plain XML file or by using Eclipse plug-in editor.

As we can see from figure 4.1, we are using four types of view extensions for our tool.

“MSMmeasurement”, “MSMInfo”, “MSMTable” and “Visualization”. The last three mentioned include in their contents the setting “allowMultiple=true”, which actually allows us to create multiple instances of these views in the IDE. Later on in this chapter we are going to speak in details about each view we implemented. We can also see how all our extensions appear in Eclipse plug-in editor in figure 4.2.

```

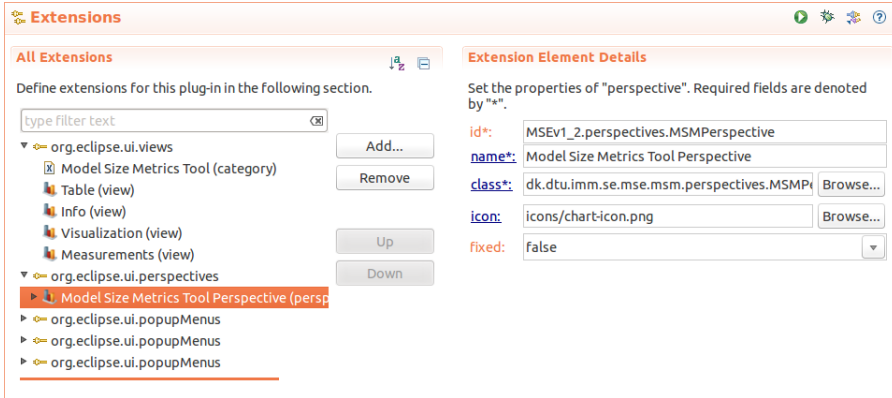
<view
  allowMultiple="true"
  category="MSEv1_2.MSM"
  class="dk.dtu.imm.se.mse.msm.views.MSMTable"
  icon="icons/chart-icon.png"
  id="MSEv1_2.MSMTable"
  name="Table"
  restorable="true">
</view>
<view
  allowMultiple="true"
  category="MSEv1_2.MSM"
  class="dk.dtu.imm.se.mse.msm.views.MSMInfo"
  icon="icons/chart-icon.png"
  id="MSEv1_2.MSMInfo"
  name="Info"
  restorable="true">
</view>
<view
  allowMultiple="true"
  category="MSEv1_2.MSM"
  class="dk.dtu.imm.se.mse.msm.views.Visualization"
  icon="icons/chart-icon.png"
  id="MSEv1_2.Visualization"
  name="Visualization"
  restorable="true">
</view>
<view
  allowMultiple="true"
  category="MSEv1_2.MSM"
  class="dk.dtu.imm.se.mse.msm.views.MSMMeasurements"
  icon="icons/chart-icon.png"
  id="MSEv1_2.Measurements"
  name="Measurements"
  restorable="true">
</view>

```

**Figure 4.1:** The views extensions in “plugin.xml” file as XML

### 4.2.3 Views

Views extend the “org.eclipse.ui.IViewPart” class. Views can be easily resized, opened, closed and moved anywhere on the workbench by the user. They are very useful for our tool since we can implement on them all the widgets needed to perform actions, present and visualize model size metrics. In our plug-in we have four views which we extending with the implementation of four classes, one for each. The figure 4.3 show a class diagram of these four classes and how they associate with eachother.



**Figure 4.2:** All extensions of our “plugin.xml” file seen by using Eclipse plug-in editor

#### 4.2.3.1 MSMMMeasurements view

From this view 4.4 we can track down all the model size metrics measurements performed by our tool. According to the “Model Size Metrics” perspective that we are using, it is placed on the middle left side of the workbench. Everytime the users request a single or multiple model file measurement.

It checks if there are any available model file paths to be loaded from the FilePath array. If there are, it uses the ModelParser class objects that get as a parameter the filepath and access their model size metrics information using the SDMetrics XMI Parser, if there are not it shows and error message. The ModelParser is actually a class created by us that uses the SDMetrics functionalities to load and transfer all those important metrics into an appropriate data structure for us to manipulate. The structure is based on six element classes that we created which store the metrics of “class diagrams”, “activity diagrams”, “packages”, “usecase diagrams”, “components” and “other diagrams” in their attributes. All these classes are then loaded into a greater class named ModelMetrics as its attributes, meaning that for every ModelMetrics objects we can have all the metrics of an XMI model file loaded ready for manipulation. The view has a level two tree and a tree viewer. The names of a single or multiple model files measured are displayed as tree nodes, and their children are the visualizations created based on those measurements. Whenever a new measurement has been performed it creates an instance of the view “MSMInfo”. Each instance has a special ID used for its identification. The metrics data structures created by the measurements are then passed to that “MSMInfo” instance for further processing.

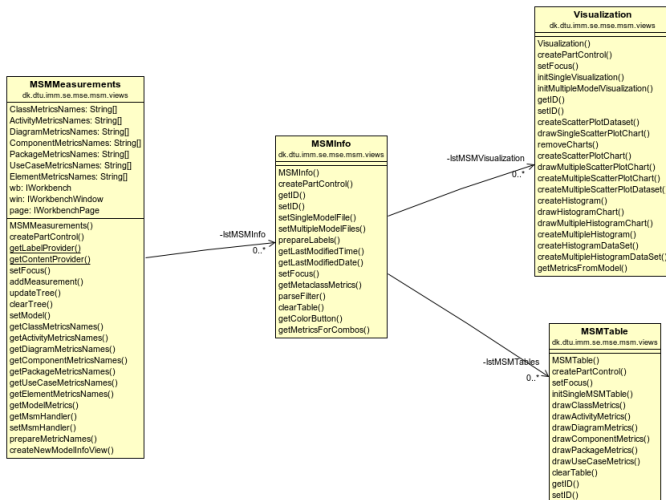
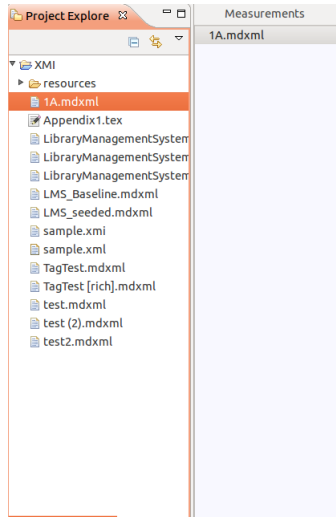


Figure 4.3: Plug-in views class diagram and associations

#### 4.2.3.2 MSMInfo view

In order to explain the functionality of the “MSMInfo” view 4.5, we must first separate into to parts. The first part that is located at the top of the view and the second part that is located at the bottom.

- The first part provides file and UML XMI related information and aswell the functionalities of loading meta-class metrics by providing a filter file in CSV format and metric exportation to files in CSV and PDF format.
- The second part at the bottom of the view allows us to specify the settings for the creation of visualizations. There is combobox with the label “Visualization Type” from which the user should select the visualization type he would like to create and be displayed. Whenever a choice is made the table labeled “Visualization Settings” provides the user with all the available settings for each visualization type. The visualization settings for example of a table view visualization is a filter combobox that we can choose the type of metrics we want to display.



**Figure 4.4:** MSMMeasurements view on the right side of the project explorer

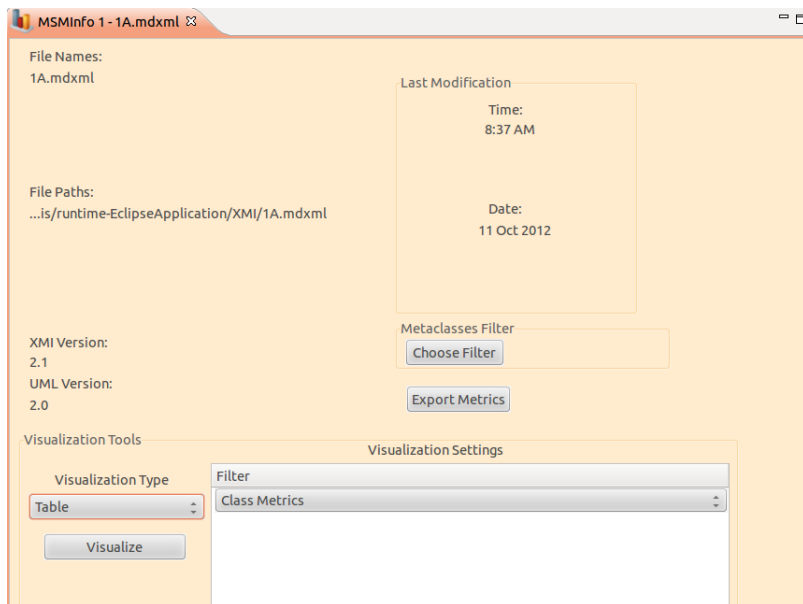
#### 4.2.3.3 MSMTable view

In this view 4.6 we can see in table format all the metrics loaded from the XMI files. This is very important since we can look more into details that we could not easily see in the visualizations.

The table has the sorting feature by using a class name `SWTTableSorter` that uses bubble sort algorithm and it as well supports scrolling to make it easy for user to explore through all of its items. On the top left corner of the view there is combobox that is visible only if we have performed a multiple model measurement, from this combobox you can specify the model file you wish see its metrics data. The view uses the Grid Layout for presenting the widgets on its parent composite. The table column names for this specific metric are created on the table and there it parsing process happening through the “ModelMetrics” class to get the data which are then added as items on the table. We can have many instances of this view and each instance has an ID attribute for identification.

#### 4.2.3.4 Visualization view

This is the view where the visualizations are drawn with the use of the `JFreeChart` library. We could not add widgets that might helped us to perform more actions



**Figure 4.5:** MSMInfo view screenshot

inside this view, because the JFreeChart library was implemented only for AWT environments and since Eclipse PDE only supports SWT, there is a constrain that allows JFreeChart to be drawn on SWT composites that only use FillLayout. That layout actually covers the whole space of the parent composite, giving us no possibility to add any more widgets on it. For that reason and since we wanted to be able to export visualizations from this view. We had to add that feature on the action bar of the view that would support visualization export when clicked. The action appears when the user presses the down arrow that appears on the top right corner of each visualization view and then chooses the action “Export Visualization”.

In figure 4.7 we can see a scatterplot that was created on the visualization view, and on the top right cornet we highlight the action for exporting it.

In figure 4.8 we present two histogram visualizations drawn in two different instances of the visualization view. We can see how cross-workbench comparison seems like and how multi model file metrics can be drawn in a single visualization. In order to understand which model file is which, in this histogram we just have bars with different colors for each one.

Name	NumAttr	NumOps	NumPubOps	Setters	Getters	Nesting	IFimpl	NOC	NumDesc	NumAnc	DIT	CLD	Opsinh	Dep_out	Dep_in
SBK.2 Information Elements.Rate	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0
SBK.2 Information Elements.Maestro	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0
SBK.2 Information Elements.Organisation	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
SBK.2 Information Elements.PersonalInformation	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0
SBK.2 Information Elements.ContactInfo	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SBK.2 Information Elements.Transaction	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SBK.2 Information Elements.Dinners Club	0	0	0	0	0	0	0	0	2	2	2	0	0	0	0
SBK.2 Information Elements.Account	5	0	0	0	0	0	0	0	1	1	1	0	0	0	0
SBK.2 Information Elements.Card	3	0	0	0	0	0	0	6	6	1	1	1	0	0	0
SBK.2 Information Elements.Agreement	4	0	0	0	0	0	0	5	11	0	0	2	0	0	0
SBK.2 Information Elements.Government	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
SBK.2 Information Elements.PrivateInformation	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0
SBK.2 Information Elements.Company	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
SBK.2 Information Elements.Customer	4	0	0	0	0	0	0	4	4	0	0	1	1	0	0
SBK.2 Information Elements.Power of Attorney	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
SBK.2 Information Elements.Preference	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SBK.2 Information Elements.Identification	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SBK.2 Information Elements.American Express	0	0	0	0	0	0	0	0	2	2	2	0	0	0	0
SBK.2 Information Elements.Private	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
SBK.2 Information Elements.Internet Banking	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
SBK.2 Information Elements.VISA	0	0	0	0	0	0	0	0	2	2	2	0	0	0	0
SBK.2 Information Elements.Dankort	0	0	0	0	0	0	0	0	2	2	2	0	0	0	0

Figure 4.6: MSMTTable view screenshot

#### 4.2.4 Pop Up Menu

The Eclipse PDE also allow us to use and adjust all the Eclipse already created useful components, such as editors, project explorers and many others. In our care we added a pop up menu extension on the eclipse’s package manager so we can improve usability for our users. And only allow supported files to be measured by the use of the pop-up menu filter setting. The pop up action is named as “Measure”.

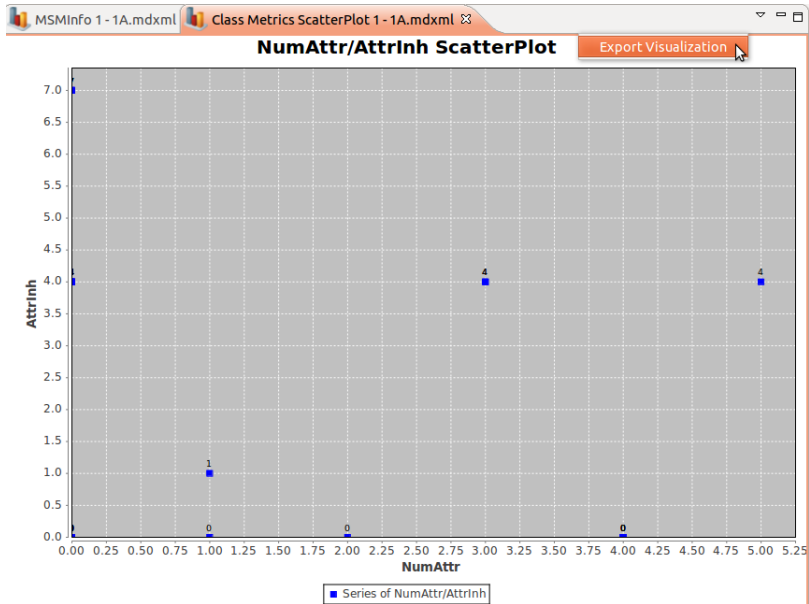
The figure 4.9 shows how the new pop up menu appears when we right click on one or multiple supported files.

#### 4.2.5 Perspective

Most plug-ins require a perspective so the user can automaticallity load, order and adjust all the components inside the IDE. A perspective is also a plug-in extension saying the IDE what components to load and where to place them. These actions are done programmatically by the perspective class that implements the interface “IPerspectiveFactory”. For our plug-in we also include a perspective to increase the usability.

Our perspective consists of:

- The project explorer: Placed on the left side of the workbench.



**Figure 4.7:** Scatterplot drew inside “Visualization” view and export visualization action highlight

- MSMMMeasurements: Placed on the right side of the project explorer.
- A placeholder: Covering the half right part of the workbench in which the “MSMInfo”, “MSMTable” and “Visualizations” instances can stack on.

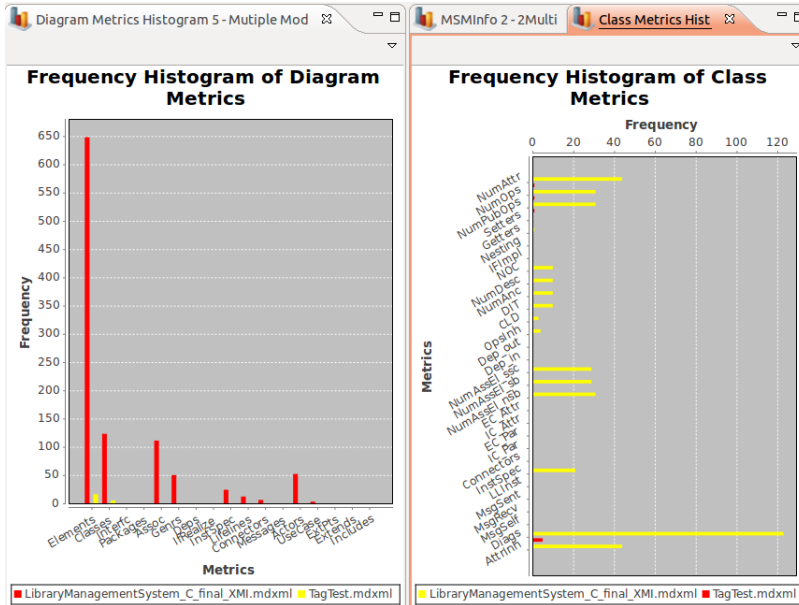
### 4.3 Libraries and components

In this section we will present the libraries and components that helped us to create our eclipse plug-in.

For the development of the Model Size Metrics plug-in we have used the following libraries and component:

- SDMetrics - Open source application for parsing XMI Model Metrics Files.
- JFreeChart - Chart creation java library using AWT and image exporting.
- iText - Library for PDF creation used in reporting.





**Figure 4.8:** Two visualization view instances for cross-workbench comparison of metrics

- Window Builder Editor - Eclipse Plug-in for easy creation of SWT GUIs.

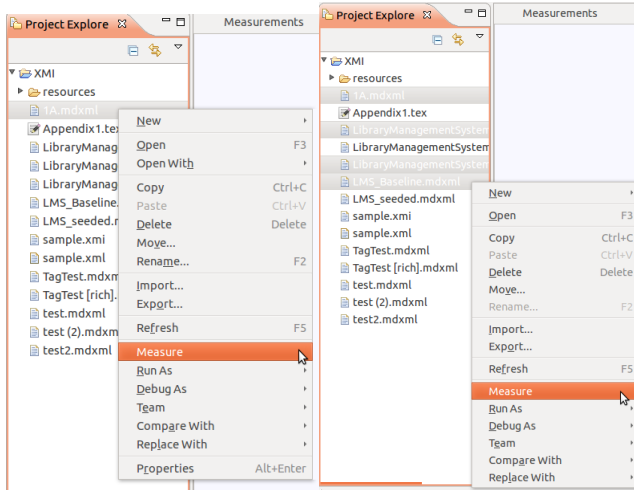
### 4.3.1 SDMetrics open core

The SDMetrics open code V2.3 we have used in this project is actually a set of java classes that comprise of[sdm].

1. The configurable XMI parser for XMI1.0/1.1/1.2/2.0/2.1 input files.
2. The metrics engine to calculate the user-defined design metrics.
3. The rule engine to check the user-defined design rules.

For our case we have used the XMI parser and the metrics engine to export the data from the model files. Next the data is manipulated by our plug-in functionalities.

The open core version comes as an extra to actual SDMetrics application which is a standalone application for model size metrics. It supports metrics and



**Figure 4.9:** Pop up menu item for measuring new single (left) and multiple (right) model files

visualizations but not in the same way our plug-in implements them, since it does not support multiple model files metrics to be loaded and compared.

### 4.3.2 JFreeChar

JFreeChart 4.10 is a free java library for chart creation. It was used by our Model Size Metrics Visualization View to present model metrics data into charts. JFreeChart supports many types of charts including scatter plots, pie charts, line charts, histograms, etc. It is a very powerful tool for chart creations as it is easy to use, free, efficient and rich in utilities. It also exists in open source version for further development. It was created to work on AWT graphics library but there is further work done to support SWT aswell[Gil08].

### 4.3.3 iText

iText is free library for creation and manipulation of PDF files. It is available in Java and in C#. We have used iText in our plug-in to create PDF files for reporting matters. The exports that our plug-in supports can be created in CSV and PDF format. For CSV plain text is used and no library was required but for PDF, iText was the solution. Inside the PDF files it enables table support and

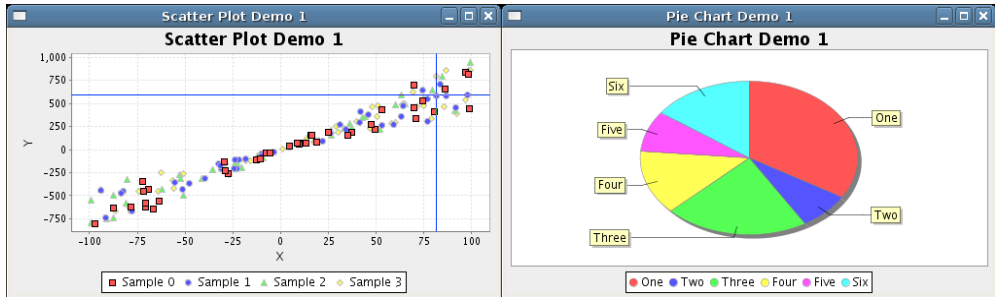


Figure 4.10: JFreeChart samples

landscape page alignment which both suits perfect for metrics reporting[ite].

### 4.3.4 Window Builder Editor

Window Builder Editor 4.11 is an eclipse plug-in for easy SWT graphical GUI design. It has helped us to create the GUIs in the views of our plug-in. It supports all SWT widgets and swings to design GUIs graphically and most coding on difficult cases such as positioning and layouts is carried out by it. Windows Builder Editor is free and it can be download through eclipse's Install New Software feature.

Its interface appearance can be seen in the following figure[wbe].

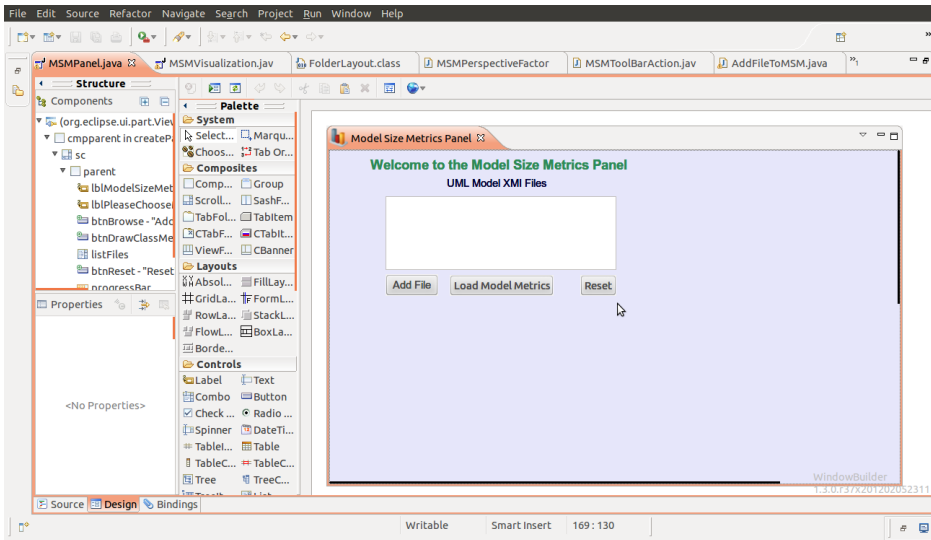


Figure 4.11: Window builder editor

# Evaluation

---

This chapter tries to evaluate our work and the tool that we have developed. We will start by passing through some different testing stages and present our findings. Then we will back to the analysis chapter take a look at the requirements we defined back then and try to see which of these requirements have been fully fulfilled, partially fulfilled or have not been accomplished. We will try to be as much critical as we can with our results since this will help us to improve and become better software engineers!

## 5.1 Testing stages

### 5.1.1 Unit testing

Throughout the phase of implementation we had to test our code in order to find mistakes and solve them. The unit testing was being done mostly in an agile way during the development. The technique we used for testing different units of the source code was by console outputs. Whenever we wanted to be sure that a method returns or an attribute has correct values, we were outputting those values and compare them with those we would expect. If the expectations were not correct then we were looking in the code for errors to fix.

One example of unit test that we performed is that we wanted to make sure that the pop up action “Measure” gets the correct file paths from the model files. We first created a unit test 5.1 inside the code which stores the filepaths in an “ArrayList” of “String” values and then we printed it out at the console of eclipse in order to see the results and check if everything is appropriate.

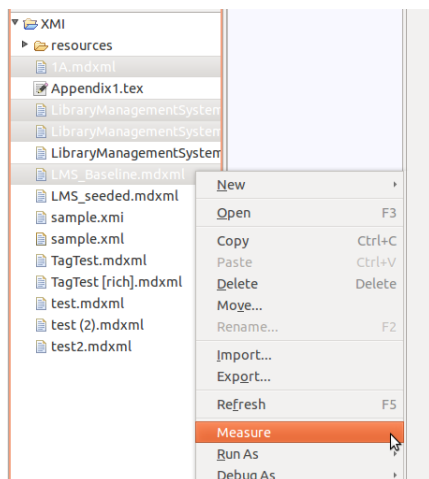
```

for(int i=0;i<list.size();i++)
{
    if(list.get(i) instanceof IFile){
        filepaths.add(((IFile)list.get(i)).getRawLocation().toOSString());
        System.out.println(filepaths.get(i));
    }
}

```

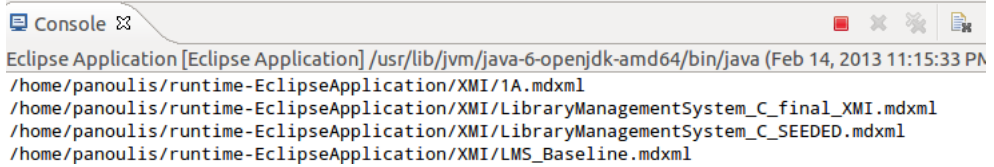
**Figure 5.1:** Unit test code for correct model file paths.

Now we had to try perform the “Measure” 5.2 action on some files which we knew their file paths and compare them with the ones that would appear at the console.



**Figure 5.2:** Unit test “Measure” action

Finally we get the results from the console output and in this test case the results seem fine 5.3. But during the development we were not getting the correct file paths because we were not calling the correct methods “getRawLocation().toOSString()” of the “IFile” object. This is a unit test example that was actually very useful to us.



```

Eclipse Application [Eclipse Application] /usr/lib/jvm/java-6-openjdk-amd64/bin/java (Feb 14, 2013 11:15:33 PM
/home/panoulis/runtime-EclipseApplication/XMI/1A.mdxml
/home/panoulis/runtime-EclipseApplication/XMI/LibraryManagementSystem_C_final_XMI.mdxml
/home/panoulis/runtime-EclipseApplication/XMI/LibraryManagementSystem_C_SEEDED.mdxml
/home/panoulis/runtime-EclipseApplication/XMI/LMS_Baseline.mdxml

```

**Figure 5.3:** Unit test console output

### 5.1.2 Functional testing

Functional testing is a form of quality assurance (QA) process. Since our tool is mostly GUI based functional testing is important for us in order to prove that it perform the way it is supposed to do. We will go through two different test cases and see if the results are appropriate.

The first case is based on the question if the metrics measured from a model file are correctly presented in a histogram chart that is being created by “JFreeChart” library. To do that, we will first see a table view in which we can see the metrics measured one by one and a histogram to see if the metrics are presented there in a good way.

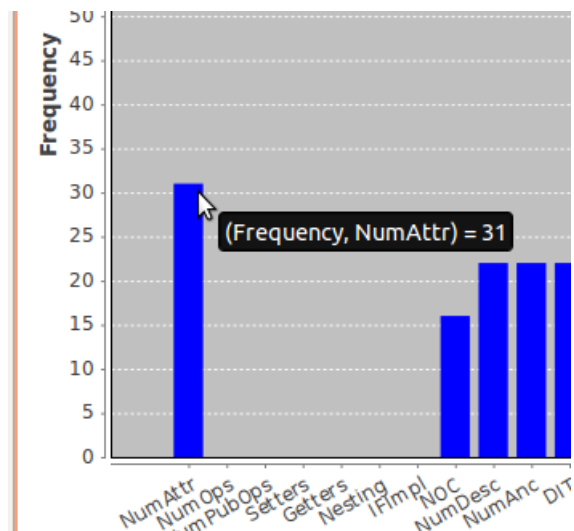
In figure 5.4 we can see the table view of class metrics of the model file “A1.mdxml”.

	NumAttr	NumOps	NumPub
	5	0	0
	4	0	0
	4	0	0
	4	0	0
	4	0	0
	3	0	0
	3	0	0
	2	0	0
	1	0	0
	1	0	0

**Figure 5.4:** Functional testing class metrics table view

From the table view 5.5 we focus on the number of attributes (NumAttr) metrics of different classes. We have sorted it in order to see only the classes that have

attributes. If we count all those number we find out that the total number of attributes of all the classes existing in the model file “A1.mdxml” is equal to thirty one (31). Now if we see the histogram chart of class metrics we will find out that it shows the same number.

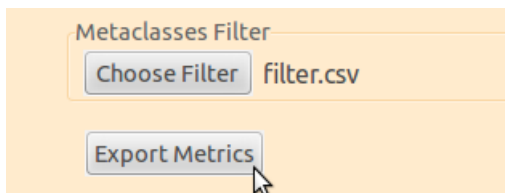


**Figure 5.5:** Functional testing class metrics histogram

So we can now say that this functional test case is passed.

In the next functional test case we will try to prove that the exportation of metrics functions as it should be. The way we are going to do that, is by trying to export the class metrics into a CSV file and see if they are actually exported correctly by comparing with the metrics we can see from the “MSMTable” view.

So in order to start we will use the “Export Metrics” button located in the “MSMInfo” view as we can see in figure 5.6.



**Figure 5.6:** Functional testing export metrics in CSV format

If we now compare the metrics from the table with the metrics exported in the





- Processor: Intel® Core™ i7-3770 @ 3.40 GHz
- RAM: 8.0 GB
- OS: Windows 7 Enterprise 64-bit
- Eclipse: Juno 4.2.1

The plug-in and its functionalities worked as they were suppose to do. But there were some differences in the way SWT widgets looked live and some minor differences in the way the perspective was placing the views inside the IDE. Since we were expecting SWT to be completely native and work the exact same way in all OS as it was stated in its specifications. We can understand that this is not exactly true.

So the second compatibility test is passed partially.

## 5.2 Requirements evaluation

Now we will have a look to requirements we set at the analysis chapter and evaluate which of them have been fulfilled and which not.

- Environment requirements: The requirement we defined concerning the environment which is the implementation of an eclipse plug-in performing model size metrics has been fulfilled.
- Tool interaction requirements:
  1. Have a project explorer to import and mentain model files has been fulfilled.
  2. Measurements and visualization viewer where we keep track of the measurements and visualizations created. And allow the easy navigation between them has been fulfilled.
  3. View components flexibilty has been fulfilled since Eclipse PDE supports that.
  4. Drag and drop (DND) support has not been accomplished.

- File import requirements:
  1. Support of different XMI and UML exists.
  2. Support for custom selection of meta-class metrics from the user is also fulfilled.
  
- File export requirements:
  1. Support for CSV and PDF exportation of metrics is fulfilled.
  2. Support for pixel and vector based image exportation is fulfilled.
  3. Support for printing visualization from the GUI is not fulfilled.
  
- Model size metrics requirements:
  1. Class metrics can be measured.
  2. Use case metrics can be measured.
  3. User specified meta-class metrics can be measured.
  
- Visualization requirements:
  1. Table views for metrics can be visualized.
  2. Single and multiple model file histograms for metrics can be visualized.
  3. Single and multiple model file scatterplots for metrics can be visualized.
  4. Single and multiple model file bubble charts for metrics can be visualized.
  5. Addition of the metrics of different model files for visualizations has not been accomplished.



# Conclusion

---

In this thesis we have managed to implement a model size metrics eclipse plug-in that can import and measure model files created by different UML modeling tools. We successfully achieved to manipulate these metrics and visualize them in different type of charts. The thesis helped us to better learn Java programming language and how to implement Eclipse plug-ins.

He had to deal with many different obstacles in every phase of the tool development. Those obstacles acquired a great deal of knowledge in the field of software engineering, and specially in Eclipse plug-in development.

Usability was very important for us since students and researches will use our tool. We believe we did a great effort in making our tool as usable as possible.

There can be improvements in our piece of software and we hope that more people will try to get into it and make it better.

We definitely believe that this thesis contributed a lot in order to become great software engineers!



## APPENDIX *A*

# Installation Manual

---

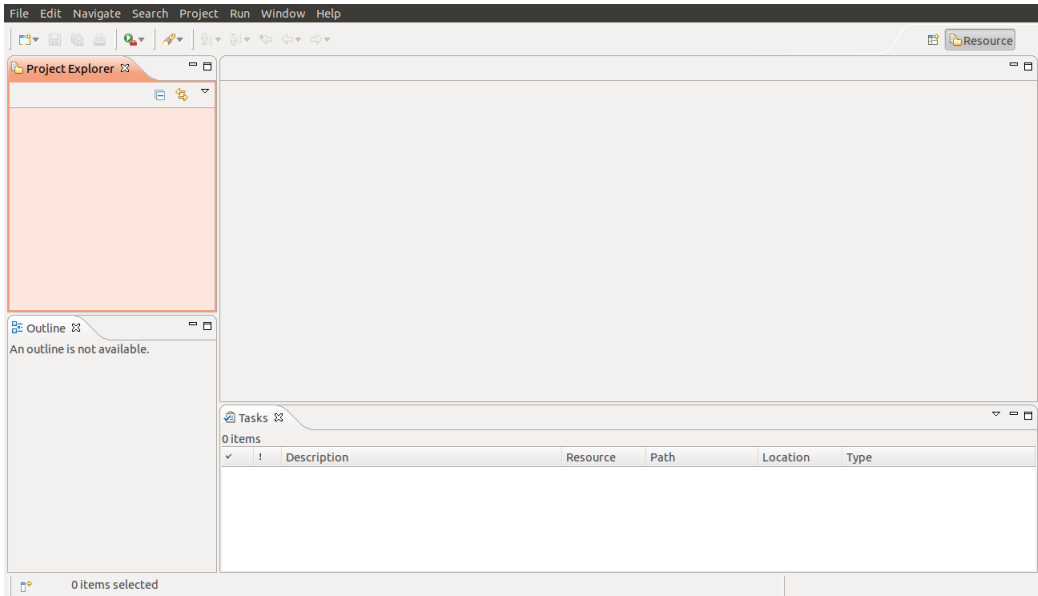
## A.1 Installation Methods

In this document we will give instruction for installing the Model Size Metrics(MSM) Eclipse plug-in that we are developing for our Master Thesis.

There are three ways for installing plug-ins in Eclipse software development platform and we will explain them using helpful screenshots in the following sections.

### A.1.1 Install as Project Repository

Let us say that we just run our Eclipse platform on an empty workspace as shown in the next figure.



**Figure A.1:** Eclipse with an empty workspace

In the next step we right click on “Project Explorer” tab and from the pop-up menu choose “Import”.



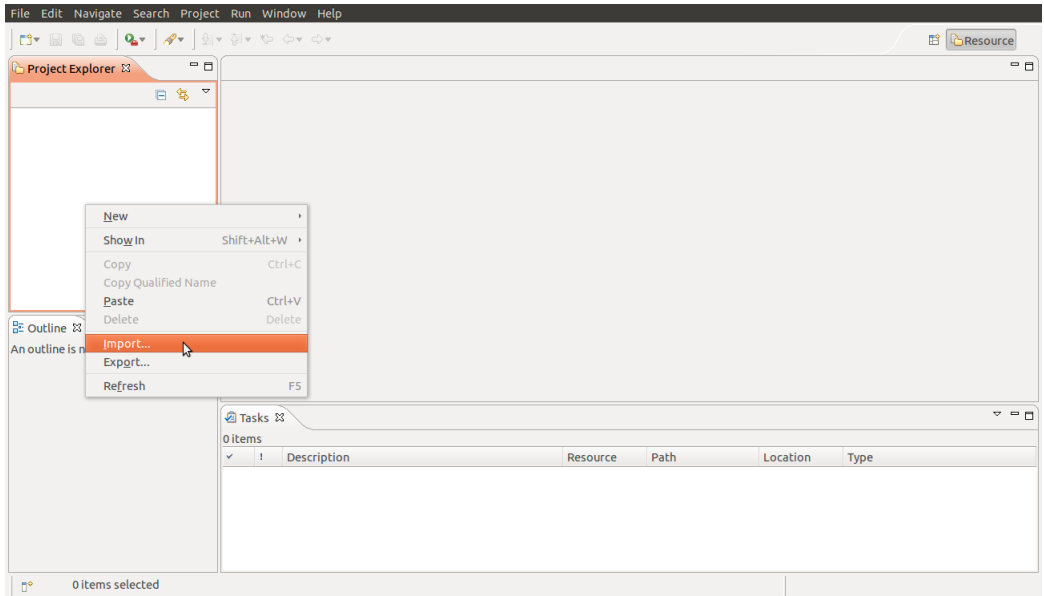


Figure A.2: Import on Project Explorer

Afterwards on the Import window that pop's up under the “General” category we select as import source the “Existing Projects into Workspace” and click “Next”.

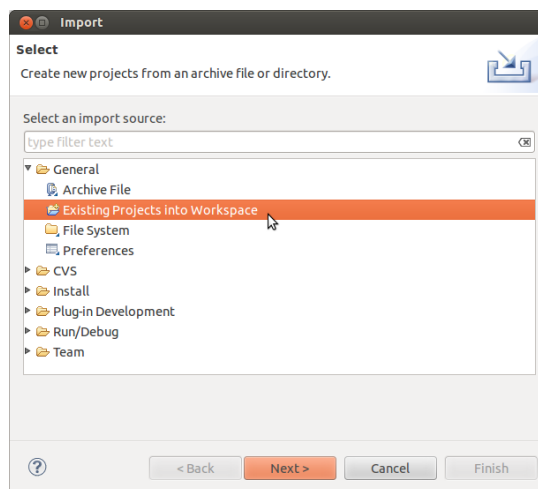
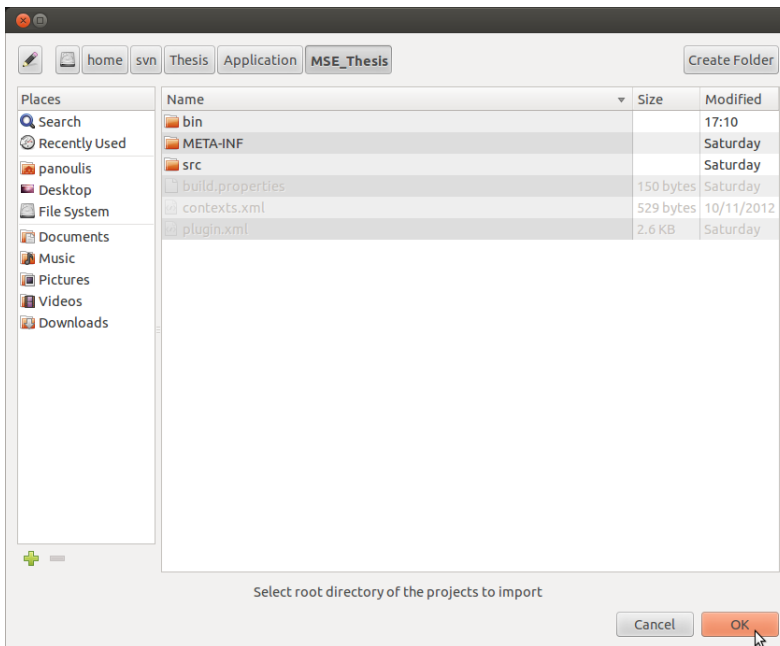


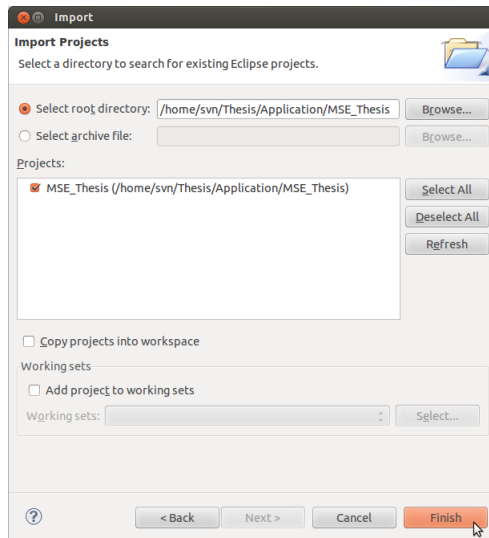
Figure A.3: Import Existing Projects into Workspace

Then we click browse and select as root directory our plug-in project which is located in our svn repository under the path “Thesis/Application/MSE\_Thesis” and click “Ok”.



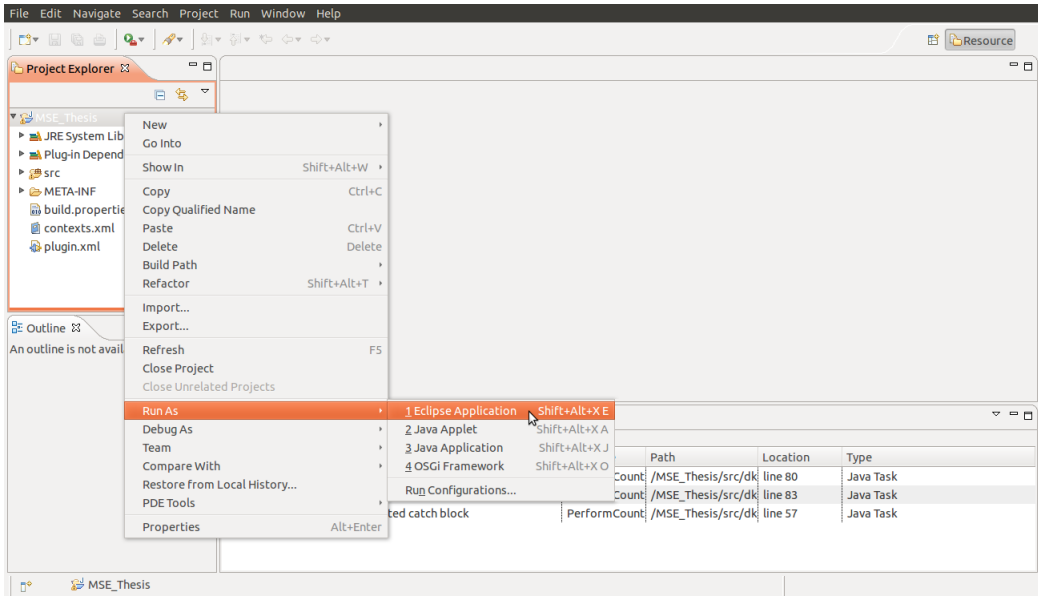
**Figure A.4:** Select Plug-in Project from svn Repository

After choosing our project the import window should be looking like the one that follows and we just click have to click “Finish”.



**Figure A.5:** Finish Plug-in Project Import

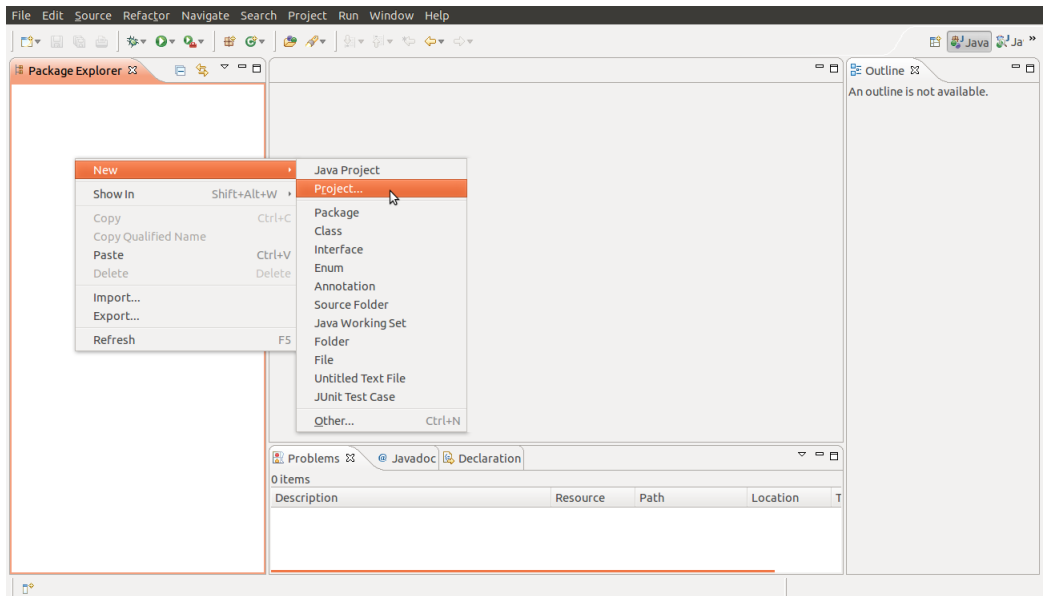
Now that we have imported the plug-in project on our workspace we can simply run it by right clicking on the project and then choose “Run As -> Eclipse Application”.



**Figure A.6:** Run As Eclipse Application

The plug-in should now be running under the new Eclipse platform that popped up. In order to perform model size metrics now we should use the “Project Explorer” which now acts as file explorer for our plug-in.

We must now right click on the “Project Explorer” and select “New -> Project”. We need this new project to import our files and perform Model Size Metrics on them.



**Figure A.7:** Create New Project

We should now create a new project under the “General” category as shows in the next Figure. Name it “Model Files” and click “Finish”.

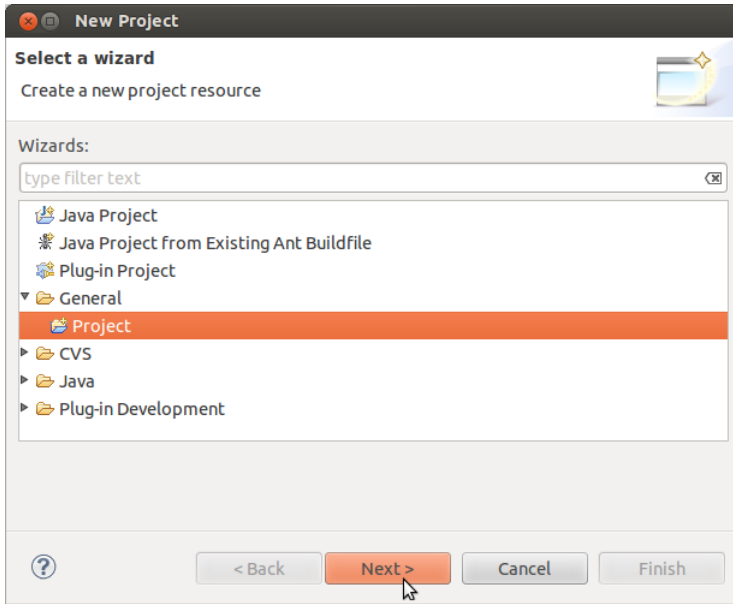


Figure A.8: Create New Project Menu

Now that the new project is ready we must import the files in it to perform Model Size Metrics. Right Click on the Project and then click Import.

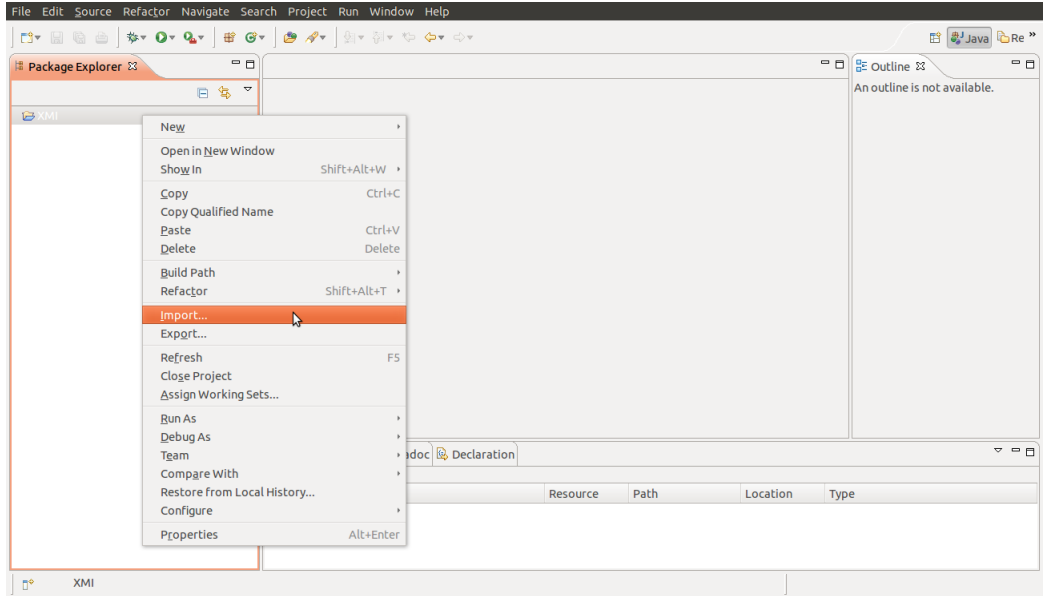


Figure A.9: Import Files on the Project

Then from the General Category choose “File System” and click next. Afterwards browse from the directory and point to the Model Size Metrics Test Files directory.

### A.1.2 Install Eclipse Plug-in manually

This way to install plug-ins is not recommended for plug-ins that are in the development phase. It is rather more usable for finished and ready to use plug-ins. In order to install an eclipse plug-in manually you need to have the plug-in project extracted into a “jar” file. Then the only thing that you should do is to place the jar file in the “dropins” folder which is located in the eclipse installation path. In my case under linux OS this folder is located at “/usr/share/eclipse/dropins”. When this is done it will automatically load the plug-in every time you will run eclipse!

### A.1.3 Install from Online Repository

This is the simplest way to install plug-ins just by pointing new software wizard to the online plug-in repository.

This method is not applicable in our case since we don't have an online eclipse plug-in repository available!



# Bibliography

---

- [CK94] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. 1994.
- [DF11] Jose Azevedo Jose Joao Peixoto Pedro Faria Pedro Silva Daniela Fonte, Ismael Vilas Boas. Modeling languages: metrics and assessing tools. 2011.
- [EC08] Dan Rubel Eric Clayberg. *eclipse plug-ins*. Pearson Education, Inc, Boston, MA, USA, 2008.
- [Gil08] David Gilbert. *The JFreeChart Class Library*. Object Refinery Limited, 2008.
- [ite] itext pdf.
- [Mar98] Michele Marchesi. Ooa metrics for the unified modeling language. 1998.
- [sdm] Sdmetrics open source.
- [spa] Sparx enterprise architect.
- [wbe] Windows builder editor.