



LINGI2252 ASSIGNMENT 2 REPORT

Group 17
Xiao XU
Xavier CROCHET

Kim MENS
Sergio CASTRO



Contents

1	Introduction	2
2	Architecture	3
2.1	Introduction	3
2.2	Overview Pyramid	3
2.3	Complexity	3
3	Comments	3
4	Naming Convention	3
5	Metrics-based analysis	4
5.1	Introduction	4
5.2	Code duplication	4
5.3	Long Method	5
5.4	Long Class	5
5.5	Long Parameter List	5
5.6	5
6	Improvements	5
7	Conclusion	5
8	Annexes	5

1 Introduction

In the first report, we try to do a manual analysis of *Glamour*. This first step gives us a global overview of *Glamour* and helps us to understand how *Moose* and his tools works. Because of the size of the system (about 270 classes), we use some tools in order to get some hints on where to begin. It wasn't exactly what was asked in the statement of the first report but now that we have all the tools in our hand, we can start looking more effectively for the needles in the haystack that *Glamour* is.

Moreover, we found some code duplication in the source code (but relatively few compared to the size of the framework) and the utilisation of the *Pattern Observer* in the *GLMPresentation* class. We thus conclude that *Glmanmour* was globally well-designed, despite the few code duplication.

The report will be divided in two parts:

1. We start by analyse the code using crieria not identifiable by metrics to see
 - How well the code is commented.
 - How good the naming convetion is.
2. Then, we develop a set of thresolded metrics we'll use later to analyse the code.

2 Architecture

2.1 Introduction

2.2 Overview Pyramid

2.3 Complexity

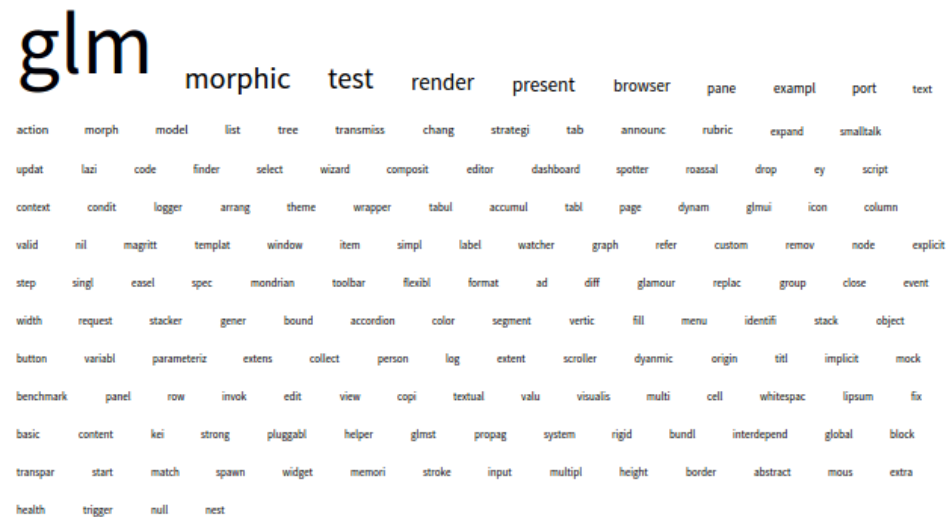
3 Comments

Commenting code is important to improve **readability** and the **reusability**. It make it easier for developers to continue a project or for student to analyse it.

There is so few comments in the code that we decide to not use a metric to analyse this part. In deed, there is only 768 comments! That's about 3 comments for each class.

4 Naming Convention

A good naming convention is also an important point in order to imprpove **readability** and **reusability** of the code. Using the *Name Cloud* utility from moose, we can display the following figure, showing the most used words **for naming classes** in the framework.



As we can see **GLM** is the suffix coming up the more often. Other suffix such as **test render** or **morphic** arise too. It seems that classes are named quite correctly in order to provide meta-information about their use.

We can perform the same analysis for methods or variable. Globally there is no bad smells coming up here. There is not randomly named classes variable or

methods. Anyway, naming convention is quite a controversial issue, so we are not going deeper in the analysis here.

5 Metrics-based analysis

5.1 Introduction

Before deciding when to refactor, we need first to identify where bad smells occurs inside the framework. Using *Moose*'s metrics set, we start by establishing criteria to indentify those parts.

In practice, we'll use the following query in order to retrieve the classes based on our thresholded metrics:

```
(MooseModel root allModels first) allClasses
  select: [:each | (each METRICNAME OPERATOR THRESHOLD)]
```

Avec

- METRICNAME - The name of the Metric
- OPERATOR - The operator for the comparaison
- THRESHOLD - The value of the threshold

5.2 Code duplication

Having the same code structure in more than one place in the system is quite nasty. The program will be better if we find a way to reunite them. The attributes we have to handle are *NumberOfExternalDuplications* and *NumberOfInternalDuplications*.

Threshold - *SmallDude*, the tool used by moose to detect duplicated code have a threshold of 3 lines. We decide here to set the threshold of the two attributes to 0, and to analyse one at the time the results.

5.3 Long Method

5.4 Long Class

5.5 Long Parameter List

5.6 ...

6 Improvements

7 Conclusion

8 Annexes