**Detection of Code smells by analyzing software metrics using Python**

Nishanth Kiran Jinagam

Bharath Reddy Nimma

Upendra Chirumamilla

Master of Science in Computer Science, Lewis University

Course: Software Architect and Design

Instructor Name: Dr. Fadi I. Wedyan

Month 12, 2022

*Introduction:*

Code smells are indications of bad design and implementation decisions, making it challenging to read and even harder to test. Despite the research community's best efforts, it is unclear how much code smells in software systems impair the testability of software. In this project, we are detecting the God class and Long Method by comparing the ck metrics for the given Java project. A prime example of a code smell is the God Class, a large and complex class that centralizes system behavior and solely uses other classes as data holders. God Classes can soon go out of control, making it more challenging for programmers to understand them, spot their problems, and add new features. There are far too many lines of code in the Long Method. Since creating code is easier than reading it, this "smell" stays undetected until the method is too long.

*Detection of God Class and Long Method:*

To detect the code smells for a given Java project, using CK metrics, we have extracted the metrics for the given project(user input), and processed these metrics and compared them using the MLCQ dataset [1].

To extract the metrics for the given project, we have used the ck metric tool [2]. We have integrated this tool [2] with our python project. This tool is packaged into a jar file(ck.jar) and can be executed using the command line using the ‘java -jar ck.jar’ command. This launches the ck metric code in a server which can be accessed in python using py4j module.

The data that is returned by the ck metric tool [2] is processed using pandas and is loaded into a dataframe. We extract the required metrics from these dataframes and process each row of data in the result with all the rows in the training dataset to find the metrics that are at a close distance to each other. In order to decide on the required metrics to classify the class or method as God class and long method respectively we have taken into consideration the following rules for classifying code smells using the software metrics:

| **Code smell** | **Denotement** | **Rule specification** |
| --- | --- | --- |
| **God Class** | GC1 | [ATFD] > 2 & [WMC] ≥ 47 & [TCC] < 0.33 [3] |
| GC2 | [WMC] ≥ 47 & [TCC] < 0.3 & [ATFD] > 5 [4] Macia et al. (2012): God class is complex, non-cohesive, and accesses more than five attributes of other classes. |
| GC3 | [NOM] > 15 | [NOF] > 15 [5] Kiefer, Bernstein, and Tappolet (2007): God Class has a lot of methods and instance variables. |
| GC4 | [CLOC] > 750 | [NOM] + [NOF] > 20 [6] Fard et al. (2013): God Class has many lines of code, or it has a lot of methods and properties. |
| GC5 | [NOM] + [NOF] > 20 Moha, Guéhéneuc, Duchien, and Le Meur (2009): God Class declares many fields and methods. |
| GC6 | [LCOM] ≥ 0.725 & [WMC] ≥ 34 & [NOF] ≥ 8 & [NOM] ≥ 14 Souza, Sousa, Ferreira, and Bigonha (2017): God Class is huge, complex, and has an extremely high number of fields and methods. |
| GC7 | [NOM] > 20 | [NOF] > 9 | [CLOC] > 750 Danphitsanuphan and Suwantada (2012): God Class has many lines of code, or it has many global variables or many methods. |
| GC8 | [CLOC] > 100 | [WMC] > 20 Liu, Ma, Shao, and Niu (2011): God class has many lines of code or is highly complex. |
| **Long Method** | LM1 | MLOC > 50 Fard et al. (2013): Long Method has many lines of code. |
| LM2 | MLOC > 30 & VG > 4 & NBD > 3 Souza et al. (2017): Long Method is huge, complex and has a high number of nested blocks. |
| LM3 | MLOC > 50 | VG > 10 Liu et al. (2011): Long Method has many lines of code or is highly complex. |

By considering the metrics used to classify the class or method into God class and long method in the above table, we have considered the following metrics to analyze if the class and method from the user’s input is god class or long method. The below table shows the metrics used in detection of their respective code smells.

|  |  |
| --- | --- |
| Code Smell | Metrics Used to for analysis |
| God Class | 'wmc','tcc','totalMethodsQty','totalFieldsQty','loc' |
| Long Method | 'wmc', 'loc', 'maxNestedBlocksQty' |

After finding the nearest neighbor for all the metrics points in the test array in the training array, all the predicted labels are appended and two csv files for God class and Long Method are generated in the Results Folder.

*Data Dictionary for results:*

File – name of the file analysed and location of the file in the local machine

Class – name of the Java class that is analysed for code smells

Predicted\_Values – (none, minor, major, critical) – if the class or method has been detected with a code smell it is classified as minor, major or critical else it is none

Wmc – weighed method count of the given class(user input)

Tcc – tight class cohesion for the given class(user input)

totalMethodQty – total method quantity for the given class

loc – Lines of code for the given class

totalFieldsQty - total field quantity for the given class

Predicted\_metrics – Metrics in the training data set that is close to this data of wmc,tcc,totalMethodQty, totalFieldsQty , loc respectively

# References

|  |  |
| --- | --- |
| [1] | L. M. a. T. Lewowski, "MLCQ: Industry-relevant code smell data set," New York, 2020. |
| [2] | M. Aniche, "Java code metrics calculator (CK)," 2015. |
| [3] | T. a. Marinescu, "Diagnosing Design Problems in Object Oriented Systems". |
| [4] | M. e. al., 2012. |
| [5] | B. a. T. Kiefer, 2007. |
| [6] | F. e. al., 2013. |