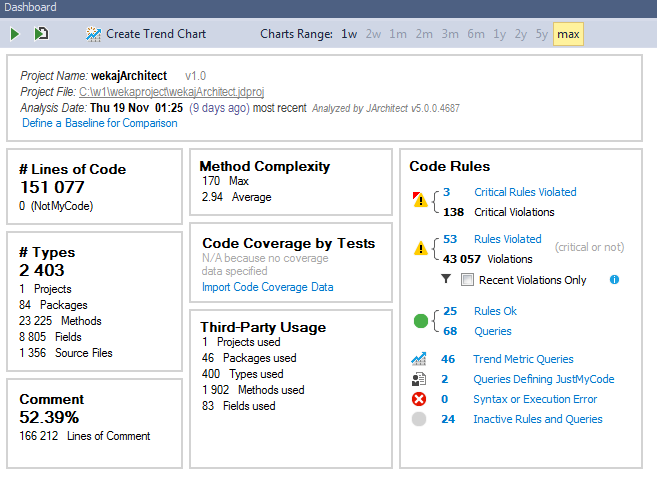
**JArchitect Analysis Report**

We have used JArchitect to analyse QA matrix of Weka project. Below are the parameters we have analyzed using Dashboard report generated by JArchitect

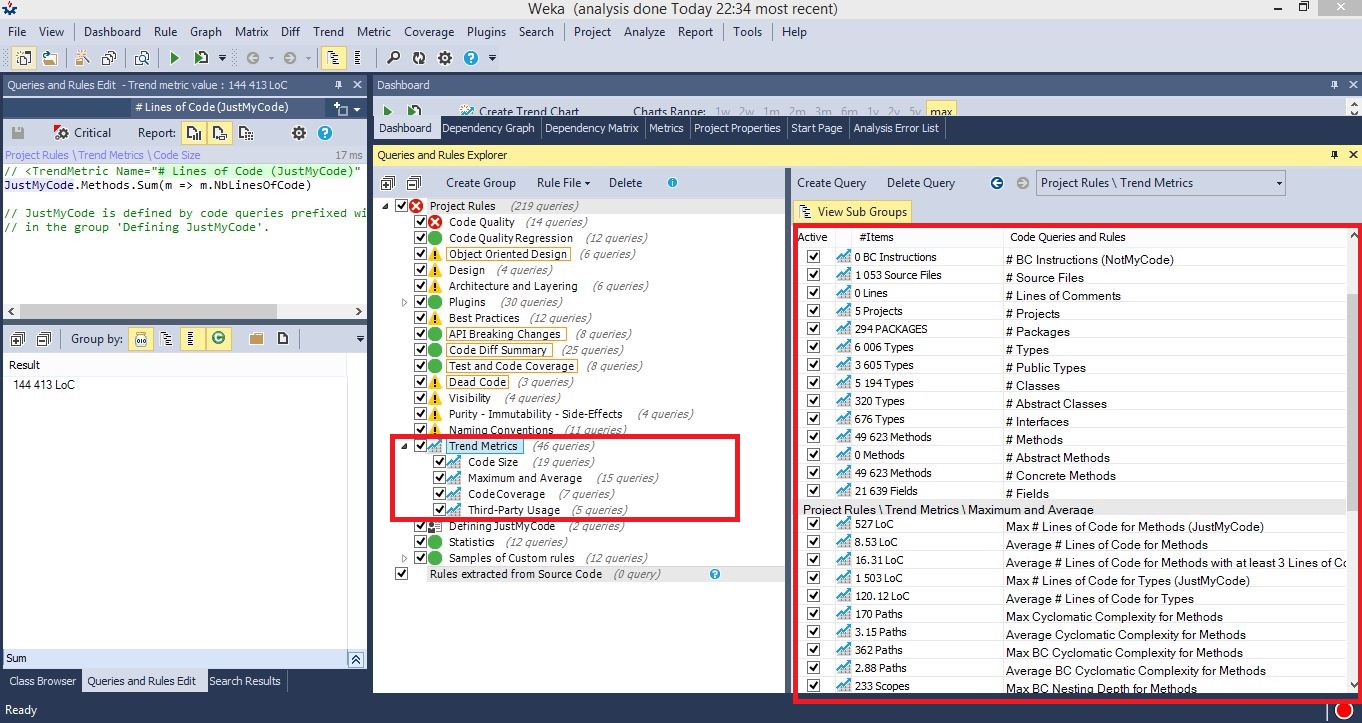


1. **Code quality:**

* The methods are too complex and big and with too many parameters. Also there are too many local variables. Also because of the usage of too many variables and Singleton instances the memory heap space drastically goes up which can be minimized .
* Static fields are initialized from instance methods which should have been avoided. Also there are some empty interfaces defined which is not a good practice.
* Obsolete types and methods are used in the code.
* Also proper naming conventions are not followed all across the objects i.e,.some instance variables are named with capital letters and names are long.
* Constructors are public, they should be private. Complex methods are not covered fully by the jUnit test cases, so the coverage of the test cases is poor.

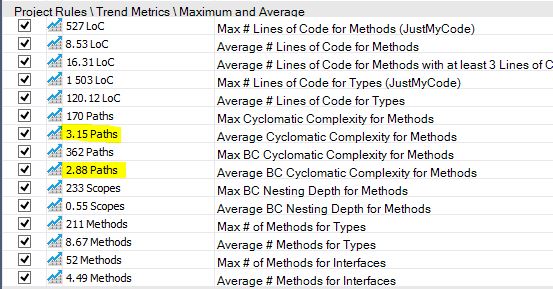
1. **Trend Matrix:**

**Cyclomatic complexity**



Cyclomatic complexity is a software metric measurement of quality assurance which indicates the complexity of a program. It is a quantitative measure of the num ber of linearly independent paths through source code. It analyses cyclomatic complexity (CC) of methods against a specified limit. The complexity is measured by the number of if, while, do, for, ?: (ternary operator), catch, switch, case statements, and operators && and || (plus one) in the body of a constructor, method, static initializer, or instance initializer. It is a measure of the minimum number of possible paths through the source and therefore the number of required tests. Generally 1-4 is considered good but it will be good to have it as minimum as possible, 5-7 ok, 8-10 consider re-factoring, and 11+ re-factor now. As low as possible, just like any other code. Generally it's recommended to keep the cyclomatic complexity below 5.

See below image, analysing the dashboard generated by Jarchitect for this project. We can see that the average cyclomatic complexity of the for methods is 3.15 paths and maximum BC CC for methods is 2.88 paths which is can be considered as very good cyclomatic complexity considering over all 49,623 abstract and concrete methods. And for maximum count of CC we can refactor the code and try to reduce the complexity.



Below is the graphical matrix report generated by Jarch, as per the CCs for respective methods as per they are marked in the image

Fig.1 Green : CC < 10 Fig.1

Fig.2 Yellow: CC < 22

Fig.3 Orange: CC < 32

Fig.4 Red: CC >50

Fig. 1

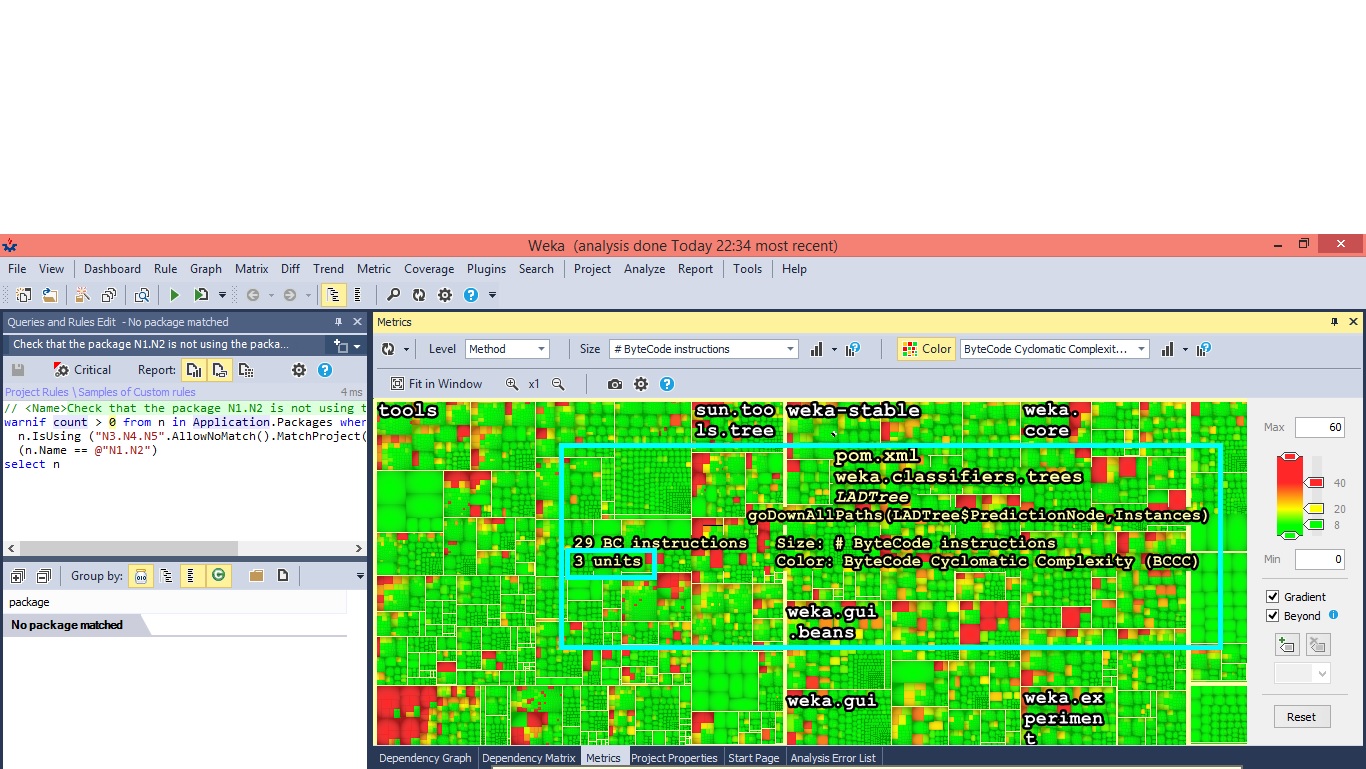


Fig 2

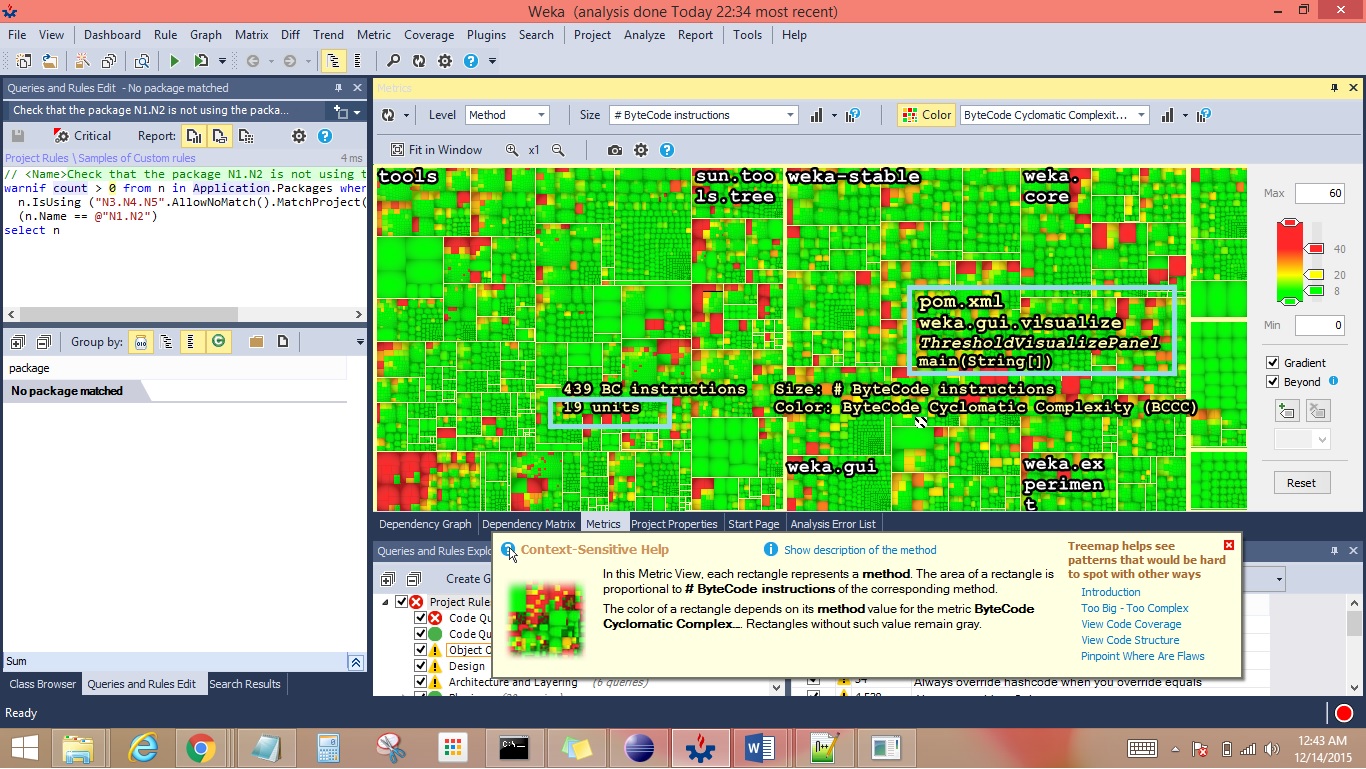


Fig 3

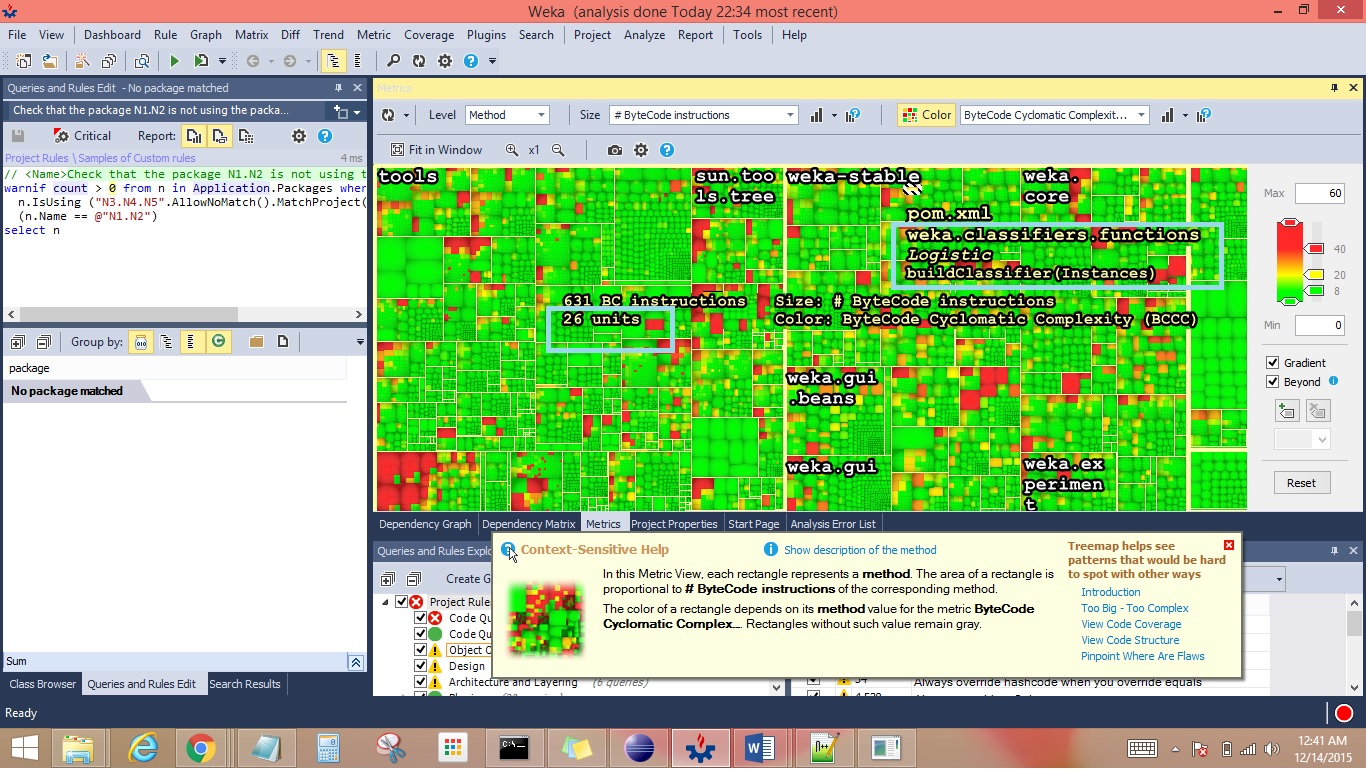
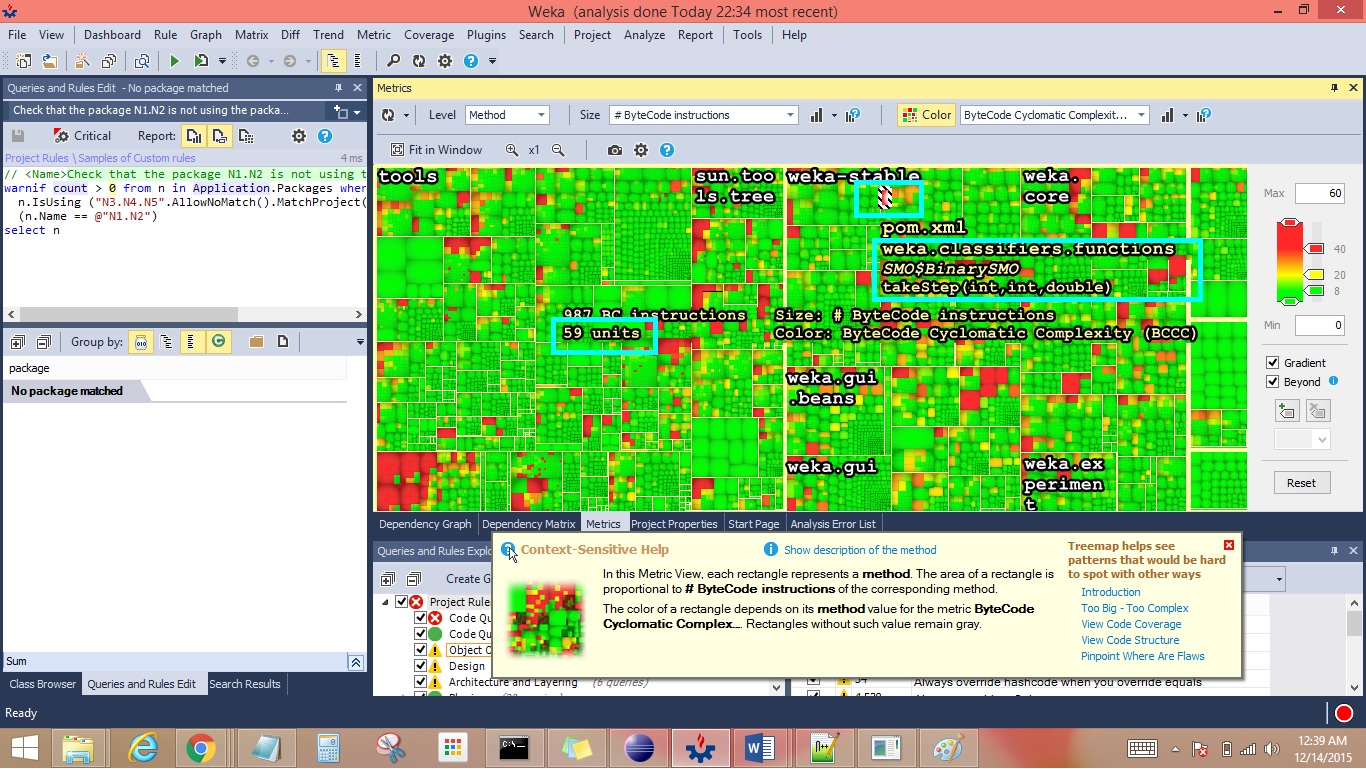
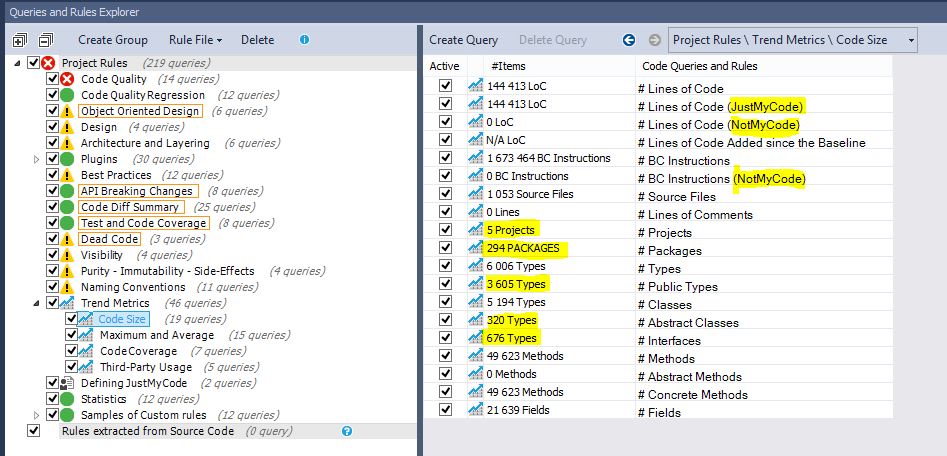


Fig 4

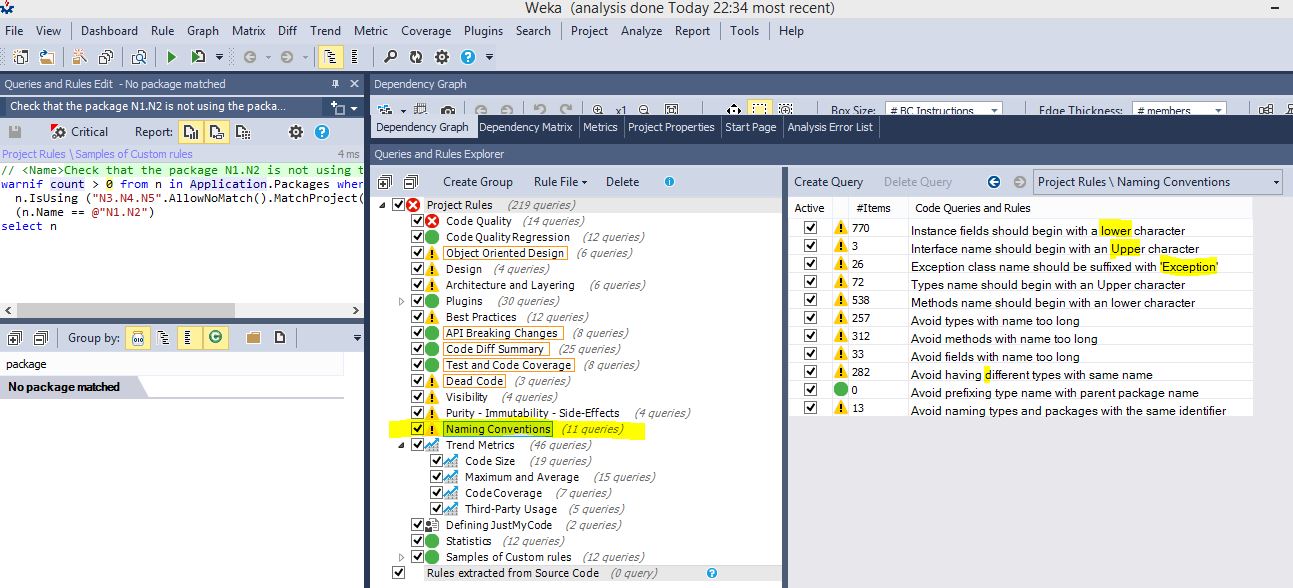


3. Below is the image containing details about matrix ’code queries and rules’. Just my code is a feature that tells the debugger to only step through the code you’ve been writing yourself and ignore frameworks and other code. The system does this by is doing this by looking at open projects, .pbd files and program optimizations. When this feature is enabled, the debugger displays and steps into user code (JustMyCode) only, ignoring system code and other code that is optimized or that does not have debugging symbols. So we can say that whole code is developed enabling this feature since there are no ‘NonJustMyCode’ no. of lines. Also in this section we can see how many global variables are used in the project, no. of packages which helps analyse the size of the project.



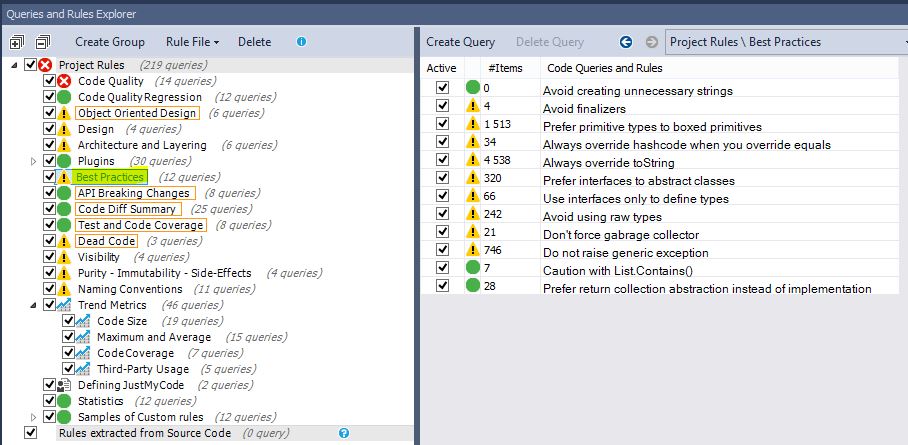
**NAMING CONVENTION:**

This seems to be the excellent parameter to analyse the naming conventions of variables, classes, methods, functions used in whole project. Below image shows the number of variables whose naming convention can be corrected.



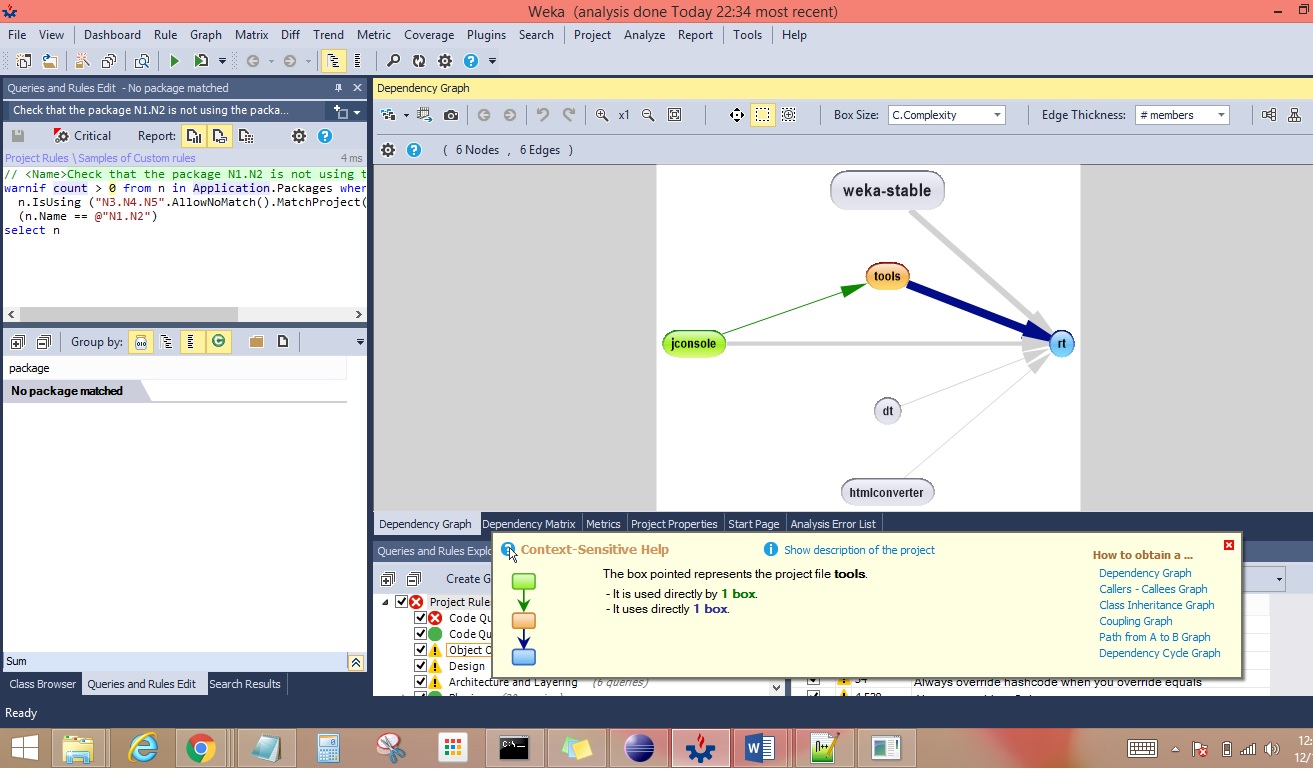
**Best Practices**

Below is the image containing information about how best practices could have been followed in this project. Jarchitect analyses this using FindBugs and .pmd.



**Dependency Graph**

Below is the image of dependency graph generated by Jarchitect when the whole weka project was built in Eclipse successfully and after importing proper jar files in Jarchitect while building the report.



FINAL DASHBOARD of JArchitect for Weka:

