REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPARISON OF SOFTWARE TOOLS FOR STATIC CODE ANALYSIS AND THE USE OF SOFTWARE METRICS

Elmira HASSANI OSKOUEI

MSc. THESIS

Department of Computer Engineering

Computer Engineering Program

Advisor

Prof. Dr. Oya KALIPSIZ

July, 2019

REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPARISON OF SOFTWARE TOOLS FOR STATIC CODE ANALYSIS AND THE USE OF SOFTWARE METRICS

A thesis submitted by Elmira HASSANI OSKOUEI in partial fulfillment of the requirements for the degree of MASTER OF COMPUTER ENGINEERING is approved by the committee on 17.07.2019 in Department of Computer Engineering, Computer Engineering Program.

Prof. Dr. Oya KALIPSIZ

Yıldız Technical University

Advisor

Approved By the Examining Committee

Prof. Dr. Oya KALIPSIZ, Advisor
Yıldız Technical University _____

Assoc. Dr. Mehmet SIDDIK AKTAŞ, Member
Yıldız Technical University _____

Assoc. Dr. Rüya ŞAMLI, Member
İ.Ü. Cerrahpaşa University _____

I hereby declare that I have obtained the required legal permissions during data collection and exploitation procedures, that I have made the in-text citations and cited the references properly, that I haven't falsified and/or fabricated research data and results of the study and that I have abided by the principles of the scientific research and ethics during my Thesis Study under the title of Metrics for software testing process supervised by my supervisor, Prof. Dr. Oya KALIPSIZ. In the case of a discovery of false statement, I am to acknowledge any legal consequence.

Elmira HASSANI OSKOUEI

İmza

Dedicated to my family

and my best friend

ACKNOWLEDGEMENTS

First of all, I would like to thank my thesis advisor Prof. Dr. Oya KALIPSIZ of the Computer engineering at Yıldız Technical University. She has been a huge support for me and very understanding and patience all this time. Whenever I had a question about my research or writing, she helped me through it very kindly.

I would also like to thank the academic staff and my colleagues in my faculty. Without their passionate participation and input, my thesis could not have been successfully conducted.

Finally, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Elmira HASSANI OSKOUEI

TABLE OF CONTENTS

LIST OF ABBREVIATIONS **VIII**

LIST OF FIGURES **IX**

LIST OF TABLES **X**

ABSTRACT **XI**

ÖZET **XIII**

1 Introduction **1**

1.1 Literature Review 1

1.2 Objective of the Thesis 1

1.3 Hypothesis 2

1.4 Related Work 2

2 Production Processes and Related Metrics **4**

2.1 Metrics for Software Purpose 4

2.2. Software Management Process Metrics 5

2.2.1 Source and cost measurement 6

2.2.2 Progress and schedule control 6

2.2.3 Product quality management 6

2.3 Software Configuration Management Processes and Metrics 7

2.3.1 Configuration tool usage metrics 8

2.3.2 Change management metrics 8

2.4 Software Design Processes and Metrics 9

2.5 Software Validation Process Metrics 10

2.6 Software Problem Solving Process Metrics 13

2.7 Software Maintenance Process Metrics 14

3 Metric Life Cycle **15**

3.1 Metric Life Cycle 15

3.1.1 Goal Setting 15

3.1.2 Metric Definition 16

3.1.3 Metric Collection 16

3.1.4 Metric Analysis 16

3.1.5 Reusability of Metrics 17

3.2 Dynamic Coding 17

3.3 Static Coding 18

3.3.1 Differences Between Static and Dynamic Types 19

4 PMD and SonarQube Tools **21**

4.1 PMD 21

4.1.1 Definition of PMD Vehicle 21

4.1.2 Using PMD 27

4.2 SonarQube 29

4.2.1 Definition of SonarQube 29

4.2.2 Using SonarQube 30

5 Case Study **33**

5.1 PMD Configuration 33

5.2 SonarQube Configuration 34

5.3 SonarQube Results 35

5.3.1 Type of Issues 35

5.3.2 Severity Level Of The Issues 36

5.3.3 Effort 38
5.4 PMD Results 39
5.4.1 Priority 41
5.4.2 Other results 42
5.5 Comparison of PMD and SonarQube 43
5.6 Tools in Contrast With the Review 46
5.6.1 SonarQube VS Review 46
5.6.2 PMD VS Review 48
6 Results and Discusion **50**
A PMD Code Results **52**
B SonarQube Code Results **117**
References **124**
Publications from the Thesis **127**

LIST OF ABBREVIATIONS

SCA	Static Code Analyze
SKC	Skills, Knowledge and Ability

LIST OF FIGURES

Figure 2.1 Retention Performance of Configuration Units.....8
Figure 2.2 Error Detection Activity.....10
Figure 3.1 Metric Life Cycle.....14
Figure 4.1 PMD Run.....22
Figure 4.2 PMD Adjustment.....22
Figure 4.3 PMD Rule Editor.....23
Figure 4.4 PMD Rulesets.....24
Figure 4.5 PMD Add Ruleset.....26
Figure 4.6 PMD Results with Custom Ruleset.....27
Figure 4.7 PMD Code Example.....28
Figure 5.1 SonarQube Analyze Result view.....35
Figure 5.2 Severity Levels Found by SonarQube.....38
Figure 5.3 Effort Time of the Issues Found by SonaQube.....39
Figure 5.4 PMD Report.....40
Figure 5.5 PMD Report Detailed about the Rules and Issues.....41
Figure 5.6 Rule Properties..... 42

LIST OF TABLES

Table 2.1 Software scope estimation and progress performance……5
Table 2.2 Structural coverage analysis……..12
Table 5.1 Type of Issues Found by SonarQube……36
Table 5.2 Comparison of CSVreader.java……..43
Table 5.3 Comparison of Distance.java……44
Table 5.4 Comparison of Server.java……44
Table 5.5 SonarQube Results Compared to the Review……..47
Table 5.6 PMD Results Compared To the Review……48
Table 5.7 Defect Results Found in the Review…… 49

ABSTRACT

COMPARISON OF SOFTWARE TOOLS FOR STATIC CODE ANALYSIS AND THE USE OF SOFTWARE METRICS

Elmira HASSANI OSKOUEI

Department of Computer Engineering

MSc. Thesis

Adviser: Prof. Dr. Oya KALIPSIZ

Software testing processes are one of the most crucial topics now because the usage of software's has grown so much bigger than before and developers are in desperate need for any ways that can help them test and evaluate their work with causing less time and lower cost. That is why software test metrics and automatic test tools plays an important role in this field. As software's are getting much bigger and more complex so as testing methods should improve.

In our study, first we emphasis on the importance of metrics used for software testing and provide samples of each type then we evaluate two well-known tools which are PMD and SonarQube that are being used often and are open source. PMD is a static code analysis tool that indicates bugs without executing the code. It uses a rule-based approach to analyze the source code. SonarQube performs automatic reviews with static and dynamic analysis of code to detect bugs.

We ran these tools on variety of Java open source codes and then measure the differences between the results.

In addition, we did a review process on some of the projects and compared the results with the tools. Our results showed that both tools have some weaknesses and also advantages. Some of the bugs were not found by the tools and their

results were not the same. The logical issues were found during the review but tools are much faster and have lower cost.

Keywords: Software metrics, automatic test tools, software testing.

ÖZET

STATİK KOD ANALİZİ İÇİN YAZILIM ARAÇLARININ KARŞILAŞTIRILMASI VE YAZILIM ÖLÇUMLERİNİN KULLANIMI

Elmira HASSANI OSKOUEI

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Danışman: Prof. Dr. Oya KALIPSIZ

Yazılım test süreçleri, günümüzde en önemli konulardan biri çünkü yazılım kullanımı, öncekinden çok daha fazla büyüdü ve geliştiriciler, çalışmalarını daha az zaman ve daha düşük maliyetle sonuçlandırarak test etmelerine ve değerlendirmelerine yardımcı olabilecek herhangi bir yol için umutsuz bir ihtiyaç içindedir. Bu nedenle yazılım test metrikleri ve otomatik test araçları bu alanda önemli bir rol oynamaktadır. Yazılımlar büyüdükçe daha da karmaşıklaşıyor ve test yöntemleri iyileştirilmelidir.

Çalışmamızda öncelikle yazılım testlerinde kullanılan metriklerin önemine vurgu yapıyor ve her türden örnekler veriyoruz sonra PMD ve SonarQube iki iyi bilinen ve sık kullanılan ve açık kaynak olan araçları değerlendiriyoruz.

PMD, kodu çalıştırmadan hataları gösteren statik bir kod analiz aracıdır. Kaynak kodunu analiz etmek için kurala dayalı bir yaklaşım kullanır. SonarQube, hataları bulmak için statik ve dinamik kod analizi ile otomatik incelemeler gerçekleştirir.

Bu araçları çeşitli Java açık kaynak kodları üzerinde çalıştırdık ve sonuçlar arasındaki farkları ölçtük.

Ek olarak, bazı projeler üzerinde bir inceleme süreci yapıp ve sonuçları araçlarla karşılaştırdık. Sonuçlarımız, her iki aracın da bazı zayıf yönlerinin ve ayrıca avantajların olduğunu gösterdi. Hataların bazıları araçlar tarafından bulunmadı ve ikisinin sonuçları aynı değildi. İnceleme sırasında mantık dayanıklı olan sorunlar bulundu, ancak araçlar çok daha hızlı ve daha düşük maliyetli.

Anahtar Kelimeler: Yazılım metrikleri, otomatik test araçları, yazılım testi.

1

Introduction

1.1 Literature Review

Software's are playing an important role in our daily lives. Nowadays, they are being used everywhere. We are becoming more dependent on them every day and they are getting more complex and much bigger than before. That is why software testing methods and tools are becoming more important. Software failures can have critical consequences. These can sometimes lead to cause huge problems just more than time and budget lost.

1.2 Objective of the Thesis

In this thesis, we introduce software metrics and the reasons why they are very important. Their life cycle and how to use metrics in the progress of the projects.

Many studies and researches have been done to help developers find bugs automatically. In this area, bug finding tools are the most developed ones. We have many tools now developed to find bugs automatically with the least cost and time possible ever. Some of these tools are open source. In our study, we examined two important bug finding tools which are SonarQube and PMD. These tools are open source and available for Java programs.

First, we look over each tool and their specifications and how they operate.

Then, we ran these tools on some projects and compare the results to see how each one can help developers and what are the advantages and disadvantages of each tool.

In addition, all the defects found were check by reviewers for some projects to observe how accurate these tools are and find out is there any defects that were not found by the tools.

This study helped us to recognize how these two tools can help developers in the software projects and what type of bugs are mostly found by them.

1.3 Hypothesis

If using the automatic software tools could prevent all the issues in the system then there was no need for any extra help and time. There are many tools developed now in this area and if one perfect tool can be designed to find all type of defects then there will be no need for any other tools. If there are some defects which cannot be found by the tools, then there is a strong need for other type of test techniques. Software developers are able to improve and add some new features for the tools and customize them according to their current project so they can increase the number of the defects found by the tools.

1.4 Related Work

There are some researches done already in this field about the automatic testing tools and their effects in software testing process. Neha Bhateja presents a study about 7 different testing tools such as Sahi, Tellurium, Windmill and etc specially for web applications [1]. Author emphasizes on how these tools can help the testing process and also the differences of automatic and manual testing. Vishawjyoti and Sachin Sharma mentioned the advantages and disadvantages of different testing techniques [2]. They used the Selenium tool in their study. As a result, they implied that the automatic software tools are the best way to improve the efficiency of testing.

Priyanka Rathi and Vipul Mehra discussed tools for automated software testing, developer oriented tools, functional testing tools and load testing tools and said the importance of metrics to analyze the quality and progress with completeness

in the product [3]. An analysis was done on automation and manual testing using software testing tools by them.

2

Production Processes and Related Metrics

2.1 Metrics for Software Purpose

Metrics are generally used to measure, track, and improve process quality [4][5]. To summarize / define metrics usage areas:

•Calculation of the time and budget required to terminate the project at a certain moment of the project

• Time and budget estimation of future projects

• Identification of the process or technology that needs to be updated or corrected

• Monitoring how the process progresses

• Use the necessary improvements to make the project more successful

• Predicting what will happen in the future

• Ensuring that the project is not out of control

These materials can be constructed according to the project, the structure of the company and the technology used. For example; the addition of the following metrics to a software test report will ensure that a reviewer will generally have an idea of the software testing process [6] [7]:

The number of test scenarios that need to be printed

• The total number of test scenarios

• Knowledge of how many successful test scenarios are successful, how many remain and how many have been discontinued

• The number of unlikely test scenarios

• How many error logs are defined and the importance of these error logs

2.2. Software Management Process Metrics

The most comprehensive of the six processes being implemented in Software Projects is the 'Software Management Process'. Planning, monitoring and supervising the acquisition, development, maintenance and support activities of the software; to make performance measures for these activities and to take corrective measures when necessary. The scope estimation and progress performances of the software used in this process realizations are classified according to the related business groups as in Table 2.1.

Table 2.1 Software scope estimation and progress performance

	Scope Prediction Performance	**Progressive Performance**
Embedded Software	75,5%	91,5%
Algorithm Design	100%	100%
Application Software	102%	103%
Software Verification	101%	108%
Hardware Interface	99%	100%
Total	90%	95%

Additional metrics determined to be used in this process are grouped under the following general headings according to their purpose.

2.2.1 Source and cost measurement

- 'Earned value' is an important metric to understand how much the project cost goes parallel to the budgeted cost. [8] [9]
- 'Schedule performance index and deviation in chart' informs the project that the schedule activities cannot be completed in time. [10]
- 'Cost performance index and deviation in cost', the performance and the deviations of the progress in cost. [10]

2.2.2 Progress and schedule control

- 'Project Variability' is to check that the metric's intended project requirements are completed according to the milestones determined, despite the incoming requests or changes requests made [11] [12]. This metric project plays an important role in cost calculations.
- 'Delayed project units', the purpose of the metric is to detect delayed or critical project activities according to the project plan and to ensure that the necessary solutions are provided for the project to be completed on time.
- 'Test case completion performance' indicates the overall performance of the test cases planned for the project.
- 'Project teams and personal performance' is a metric showing job completion performance of teams and individuals who are responsible for different parts of the project. Motivation, team fit, production, etc. in personal performance evaluations are influential. [13]

2.2.3 Product quality management

- 'Documentation quality & production performance' is a metric showing the document production performance and document quality of the project.

2.3 Software Configuration Management Processes and Metrics

The process in which configuration tasks are executed in software projects is the 'Software Configuration Management Process'. The aims and activities of this process can be summarized as follows;

- Identification of the software products and the configuration of each software product,
- Storage of the information necessary to prepare the instructure and environment of the project specific configuration management system and to bring the product back.
- "Configuration bases" are taken at certain points in time and the integrity is preserved,
- The changes made to the configuration units can be under control and reported,
- The ability of the software to be re-used in case of similar or common, the efficiency of the projects is increased by sharing information.

The graph of the metric showing the retention trends of the configuration units used in this process execution is as shown in Figure 2.1.

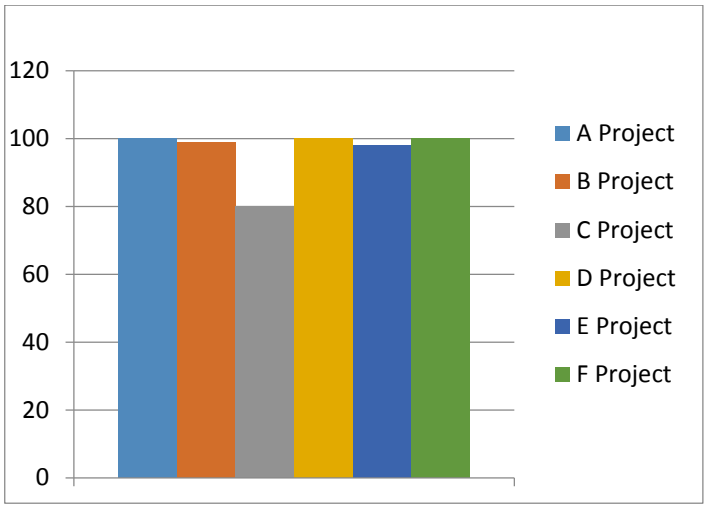


Figure 2.1 Retention performance of configuration units

Additional metrics defined for use in this process are grouped as follows:

2.3.1 Configuration tool usage metrics

- 'Configuration management information pool growth rate,
- Baseline acquisition performance (frequency and speed) shows baseline acquisition policy and performance through a Confirming management tool.

2.3.2 Change management metrics

- The 'ratio of change activities to ClearQuest individual change requests' shows how well they respond to incoming change requests.
- 'Code line size and variation' indicates the tendency of the progeny to grow in line with the code line. [12]

2.4 Software Design Processes and Metrics

This process aims at designing and developing the software units needed within the scope of product design and development and to create the design documents. The main metrics used in this process are as follows:

1) The objectives of the software-derived error performance metric during the ongoing integration testing;

a) Software in software proficiency tests test case where the number of errors originating from should be less than 5%.

b) The ratio of the number of errors found by static code analysis to the number of developed codes must be less than 0.05%.

2) Implementation re-usability metric measures two basic things; reused is the project component number, and reusability is the total number of components developed in consideration. The number of components targeted for reuse varies according to the project. Hence, it can be decided after the number of revised, semi-edited and unmodified modules has been determined to metric norms. [14]

The main additional metrics that can be used in this process are determined in this way;

- 'Design (class, package, subsystem) progressive performance' provides information on the number of structures realized in project design and sheds light on the developing size of the project. [15]
- 'Linkage level' is the quality of project design the modules that are similar to each other to determine the amount of similarity by dynamic code analysis targets.
- 'Compliance level' refers to the degree to which the project modules contribute to achieve a single, well-defined goal. When measured at the class level, this metric shows that all elements of the class are strongly

correlated. The measurement is made according to 3 types of compliance level and criteria; between methods, between classes, as a genetic fit. [16]

- The 'level of polymorphism' is also necessary when measuring the presence of a design in which there is a common interface between the various types of objects, and the final decision on the amount of reusability. [17]
- 'Exception handling' measures the degree of error or exceptions encountered in the software. In doing so, the 'catch' block numbers in the class are considered. In addition, the metrics software's robustness to application errors, and the ability to continue working without crashing despite small flaws can be measured. [18]

2.5 Software Validation Process Metrics

This process is to determine that the software products meet the requirements, the standards, and identify possible errors in phases that are easier to recall. The main metrics achieved are as follows:

1) The target of the error detection performance indicator metric is; errors in the software validation process, quality testing, acceptance testing and software validation process, the ratio of errors occurring in the sum of activity should be greater than 95% data as Figure 2.2;

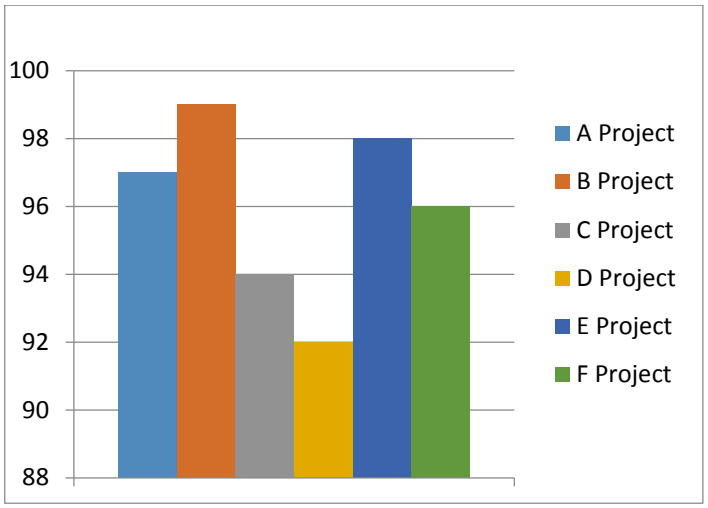


Figure 2.2 Error Detection Activity

2) The completion status of the tests being implemented is the target of the performance indicator metric; the planned schedule for the tests should be greater than 80% of the actual duration.

The main additional metrics that can be used in this process are as follows;

- 'Functional point coverage performance' is a measure taken to ensure that some functional points in the request are covered by the code and ensures that future functionalities are covered by the functional points that may cause problems, and also the functional point counting and identification at the beginning of the project illuminates the functional size of the project.
- 'Structural coverage analysis' shows how much of the project code is tested and how much of the code is passed through the tests [12]. See Table 2.2 for detailed explanation and evaluation.

Table 2.2 Structural coverage analysis

Metric Name: Structural coverage analysis
Development Strategy: It shows how much of the project code has been tested. In future it will also show how much of the code needs to be tested.
Norms: Structural Coverage Analysis Percentage of Performance: (KS - (OK + TK + EI + MA + DC)) / KS% => 100% ** For DO 178-B standards a structural test coverage analysis is expected to be close to 100% for success. KS: Total number of lines of project code. OK: The number of lines in the code that are considered "dead code". TK: The number of lines of code that the test does not cover EI: The incompatibility and the corresponding code line that is missing in the code due to missing or missing. MA: The number of lines of code that must be manually tested DC: "Defensive" Code line count *** Automatically covered vehicles are used. (Eg LDRA or G-Cover)

- 'Inspection frequency and results' are very important in terms of project supervision and give a clear idea about project progress when comparing the results with the project plan [15].
- 'Project error criticality and importance sequence' ensures that we monitor the criticality ratings of errors in the project and identify important errors

that may occur. We can detect the weak and problematic part of the project [12].

2.6 Software Problem Solving Process Metrics

This metric measures maintenance, testing and etc. related to software products that are under the control of this process purpose configuration to identify report, follow up and solve all the problems that may arise in phases.

The main metrics used in this process are as follows:

1) The target of the average closing time metric of the error reports in use is as follows; the average closing time of closed error reports should be less than 30 days.

The main additional metrics that can be used in this process are as follows;

- 'Error reports quality and time spent' on the project gives information on the quality of the reports of the faults found and the effort made for them.
- 'Fault-tolerant performance' gives information about how quickly the errors in the project are removed and how much of the planned fault-off time limitations are met [12].
- 'Fault category and intensity' gives information about the types of faults detected and places where they are concentrated. Density measurement is done by proportioning the number of incoming errors to the number of lines of code. For example, the error count for 1000 line codes should not exceed 7 years [12]. It is a metric that is often used in order to take precautions and make mistakes that can put the project in trouble.

2.7 Software Maintenance Process Metrics

The purpose of this process is to ensure that the integrity of the software and its reliability is maintained in a consistent manner when the balance is requested by the user in the current software product or in the work environment or when a change is foreseen by the software maintenance officer.

The main metrics used in this process are as follows:

1) The target of the average open time frame of the ongoing change requests is as follows; the software maintenance process ensures the desired performance for projects whose average turnaround time for change requests is over 1 month.

2) The target of the 'return results and performance from the user related to the closed problem' being applied is to stay above 95% in the satisfaction survey conducted at the end of the project and to ensure customer satisfaction. The main additional metrics that can be used in this process are as follows.

- 'Error correction performance and effectiveness' means the speed with which error reports are removed in the maintenance process and how quickly the system can be removed again [12]
- 'Added and changed code quantity and effectiveness' shows how the amount of code added to or changed in the main code quantity during the maintenance process affects and requires the system.
- 'Pending workload management and realization performance' shows how well the amount of pending work is managed in the process of consideration [19].

3

Metric Life Cycle

3.1 Metric Life Cycle

Many experts working in the technology sector are aware of the importance of metrics, but are having difficulty deciding where and how to start [4]. The metrics have a life cycle like software and need to be developed in this cycle. Since metrics are also objects that are living, they need to be collected under a consistent approach. For this reason, the definition and application of a metric life cycle is not a necessity but a necessity for healthy effort estimation. [20] This cycle of life can occur from water steps. (See figure 3.1)

Figure. 3.1 Metric Life Cycle

3.1.1 Goal Setting

The answer to why should the metric be collected is determined at this stage. The Work which is done without the intentions means loss in both time and budget. There will be also no efficient work. If the goal is determined, the route will be easier and the target will be more effective and accurate.

For example:

- More test runoff for risky areas
- Increase customer satisfaction and test satisfaction by 20% within 3 years
- Estimating test effort for future projects
- The level of importance of mistakes to return to the customer is not at a critical level

3.1.2 Metric Definition

There are hundreds of metrics in the market. All of this will mean collecting is inefficient and spending the limited test time in the void. Metrics for the intended purpose should be defined and studied. Selected metrics should be easy to collect, reportable, and statistically be able to model. Projects, needs, technology used and customers' needs may constantly change. According to these, the selected metrics must be up-to-date and the metric collection process should be kept alive. Selected metrics should answer questions about defined goals [21].

3.1.3 Metric Collection

Using test management tools during run-time and reporting of tests can greatly facilitate metric collection. The results of metrics determined by appropriate filters can be collected. Aggregated metrics and top-level metrics can also be computed using simple mathematical operations.

3.1.4 Metric Analysis

Studying the progress of the project over the metrics held at specific times of the project will help to reduce the risk level of the project. The immediate intervention of the necessary areas to deal with the managers in the project and to intervene immediately can lead to very serious losses. As an example, if the test plan pressure test scenarios increase the error density, a very serious number of errors may be expected in the vendor version. In this case, it may be considered to

increase the number of test scenarios to be tested for impact analysis after error resolution or to make test scenarios more visible and efficient [22].

3.1.5 Reusability of Metrics

The collected metrics need to be stored in an understandable way to be persistent and usable in future projects. Metrics must have a high degree of permanence and accuracy when it is assumed that the aggregation purpose of the metrics is to be used during the test effort estimation requirement. If you think you will not only use these metrics, it is very important to leave an understandable and easily applicable metric set by other stakeholders [23].

3.2 Dynamic Coding

A mechanism called CodeDOM (CodeDOM), which provides developers with source code-spreading programs to create source code at run time in multiple programming languages, is based on a single model of code document object code represented by the .NET Framework to be handled.

In order to represent the source code, some other source code structure is connected to the CodeDOM elements so that the Models CodeDOM graphic will form a known data structure.

System.CodeDom The Namespace defines the types of code that can represent the logical structure of a particular programming language independent source code. System.CodeDom. Compiler The Name field defines the types of code that are used to create CodeDOM charts and to manage compilation of supported source code on the fly. You can expand supported language pack compiler vendors or developers.

Modeling independent source code on the fly can be useful when a program needs to create source code for an ambiguous target language or multi-level program model. For example, if the language support is available in CodeDOM, some

designers use CodeDOM as a language abstraction interface to create the source code in the correct programming language.

The .NET Framework includes code builders and code builders include CSharpCodeProvider, JScriptCodeProvider, and VBCodeProvider.

3.3 Static Coding

The cost of using the tool for static analysis of software source code is a good thing. There are a number of open source and commercial tools that examine one or more of the topics specified during the analysis. There is a benefit in choosing the widest tool for comprehensive analysis. However, complementary tools can also be used. Because of the wide range of topics that can be examined statically, some companies offer a tool set instead of a single tool.

Some of the existing SKA tools examine the source code of the software textually and do not need any extra information [33]. More comprehensive analysis tools require platform and compiler information to be used. [31, 32].Compiling and analyzing the source code allows for more reliable and detailed results [29, 30].

By requiring the establishment of a special test environment, the software development process can be easily implemented in every step before packaging, and it will be possible to maximize the integration of each company in the development process to the best of its own. The adoption and effective use of the SCA is not dependent on the acquisition of the vehicle but requires a good strategy and organizational orientation [28]. Therefore, in the process of incorporating the SKA into the software development process, it should be determined who will use the SKA, which process steps will be used, and how the results will be evaluated.

3.3.1 Differences between Static and Dynamic Types

Especially in the last 10 years many programming languages have been developed. An incredible number of programming languages are designed and new features are added to them.

The software sector is growing as technology continues to evolve and today's needs continue to grow. One of the tools used in software development processes is programming languages. Since a Developer does not know all the programming languages, adaptation to different programming languages used in different projects can be a bit difficult. The purpose of programming languages is not changing; to be able to explain to computer. The only difference is the ways in which software is developed and the methods used. This is why the programming languages are separated at some point. One of these points is Type Systems.

When we start to examine a programming language, we first get information about type issues. For example; Java Static Type Checking or Python Dynamic Type Checking. This information plays a major role in grasping and developing our language.

Type systems in programming languages group values in certain types. The same values are grouped under the specified type name, allowing software developers to use the data more effectively and meaningfully. It is very important that the data is processed efficiently and without error before it is processed. Operation on unrelated data is avoided by this type system. For example; An Integer and a Boolean value are used in the same expression, and a meaningless value is avoided by type systems.

Type checking of the data is done to prevent mistakes before each processing in programming languages. Such checks are called "Checking". Checking is done at compile-time (during compilation) or run-time (during runtime). This difference is one of the main divisions of the Static and Dynamic Typed languages.

In statically typed tongues, each variable must be a constant type. This type is determined by the developer or by the compiler. Static Typed tongues are made Type Control prior to operations, so the types of operands must be specified. Type checks are done during compile. Examples of such languages are Java, C, C ++, C #, F #, Ada, Fortran, Haskell, ML, Objective-C, Pascal.

In dynamic type languages, values have a constant type. However, variables and expressions do not have to have a certain type. During each operation, the operands can produce different values. Types of variables and expressions are determined during run-time. Examples of such languages are Groovy, Javascript, Lisp, Lua, PHP, Prolog, Python, Ruby, Smalltalk languages.

Static Typed Tongues and Dynamic Typed tongues are putting forth many ideas about the superiority of each other. They both have points that are superior to each other.

In static typing languages, type checking is done during compile-time, so there is no extra type checking during program execution, which means that the application is much more efficient and fast to work with. In dynamic type tongues, control during run-time causes a slight or even slowdown. Also, because of this feature, the type information of each variable has to be stored in memory. This means an extra memory load. There is no need for an additional memory as the type is checked during compile-time in static type tongues.

If there is no error during compile-time in static type languages, type error does not occur. That is to say, dynamic type is much more secure than languages. There is no such certainty in dynamic type tongues. Dynamic Typed languages, on the other hand, provide great flexibility to developers. It is very useful in applications where the incoming dataset is unknown. In addition, application developers can assign different values to different variables at different points.

4

PMD AND SONARQUBE TOOLS

4.1 PMD

In this section, all the details of PMD tool and the usage of the tool is described.

4.1.1 Definition of PMD Vehicle

PMD is an open source code analysis tool written for java.

It can be downloaded from http://sourceforge.net/projects/pmd/files/. In this way, JDeveloper, Eclipse, Netbeans and various IDEs can be downloaded for the appropriate. I downloaded Netbeans version for working on Netbeans. Then I installed the IDE on the plugin. I did this by following Tools> Plugins> Downloaded> Add Plugins. .> pmd.nbm. After installing the plugin, if I right click on a project, it can be run from the Tools section by running Run PMD or by pressing Ctrl + Alt + P. (Figure 4.1).

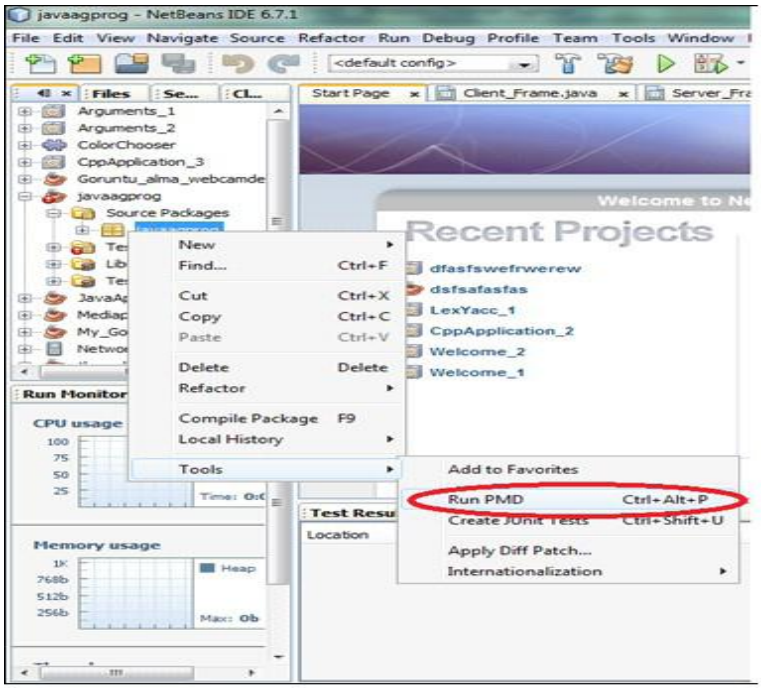


Figure 4.1 PMD RUN

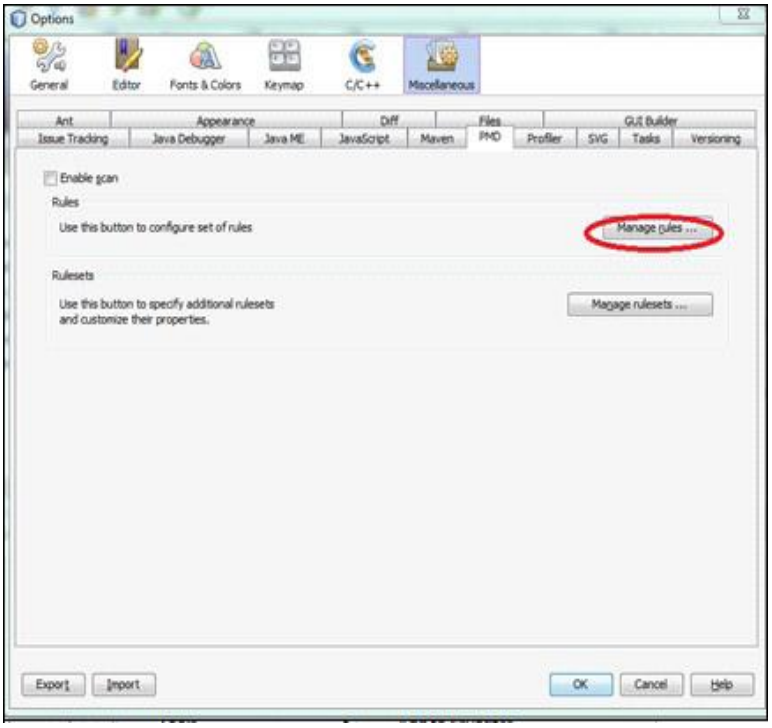


Figure 4.2 PMD Adjustment

It is also possible to adjust what PMD will look at when analyzing the code. Accordingly, if you look at the following features of the code, there is no need to look at the following can be made. At the same time, it is possible to store and reuse the old settings and use different RuleTets for different scenarios. First open the Tools> Options> Miscallaneous> PMD window (Figure 4.2).

The "Rule editor" tab appears when "Manage Rules" button is clicked. Here rules are determined by the user. The ">>" key can also be used to load all rules. See figure 4.3.

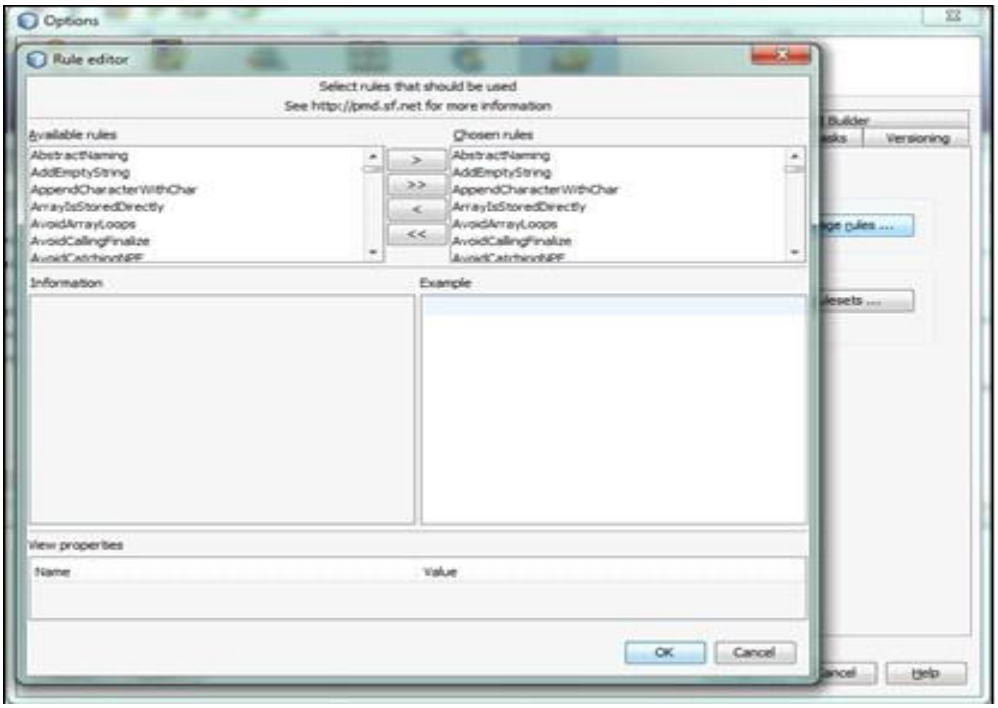


Figure 4.3 PMD Rule Editor

Once we have set the rules in this way, we can now test the code. I selected the "Multi-Server Client" program that I wrote during the summer internship for testing. This program is a program that allows computers to communicate over the network and allows the creation of multiple servers and clients on the same computer. PMD issued 128 warnings when Run PMD was executed. (see Figure 4.4). One of them was the EmptyCatchBlock fault. There is an explanation beside

the fault. A more detailed explanation and example of the code is available when an error is selected in the "Rule editor" window. The reason why so many warnings came out was that I made all the rules valid.

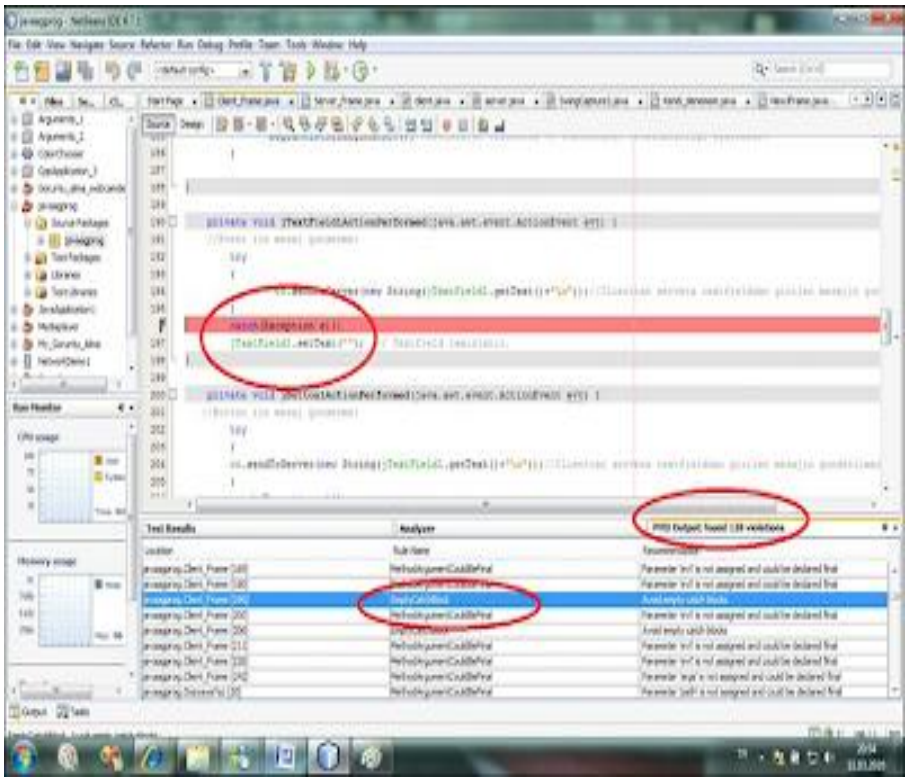


Figure 4.4 PMD Results

The use of a previously prepared Ruleset is possible. This time we press the Manage rulesets button in the Options Window. Add ruleset button to load an xml file (see Figure 4.5). We can change this xml file by ourselves, we can edit it according to our needs. Or, we can use a ruleset previously written by others. Below is the inside of the Ruleset I'm using.

```
<?xml version="1.0" ?>
- <ruleset name="Custom ruleset" xmlns="http://pmd.sf.net/ruleset/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pmd.sf.net/ruleset/1.0.0
http://pmd.sf.net/ruleset_xml_schema.xsd"
xsi:noNamespaceSchemaLocation="http://pmd.sf.net/ruleset_xml_schema.xsd">
<description>Imagine PMD Rule Set</description>
<rule ref="rulesets/basic.xml" />
<rule ref="rulesets/codesize.xml" />
<rule ref="rulesets/clone.xml" />
<rule ref="rulesets/coupling.xml" />
<rule ref="rulesets/finalizers.xml" />
<rule ref="rulesets/imports.xml" />
<rule ref="rulesets/logging-java.xml" />
<rule ref="rulesets/logging-jakarta-commons.xml" />
<rule ref="rulesets/naming.xml" />
<rule ref="rulesets/optimizations.xml" />
<rule ref="rulesets/strings.xml" />
<rule ref="rulesets/sunsecure.xml" />
<rule ref="rulesets/unusedcode.xml" />
- <rule ref="rulesets/codesize.xml/CyclomaticComplexity">
- <properties>
<property name="reportLevel" value="8" />
</properties>
</rule>
- <rule ref="rulesets/strictexception.xml">
```

<exclude name="**SignatureDeclareThrowsException**" />

</rule>

</ruleset>

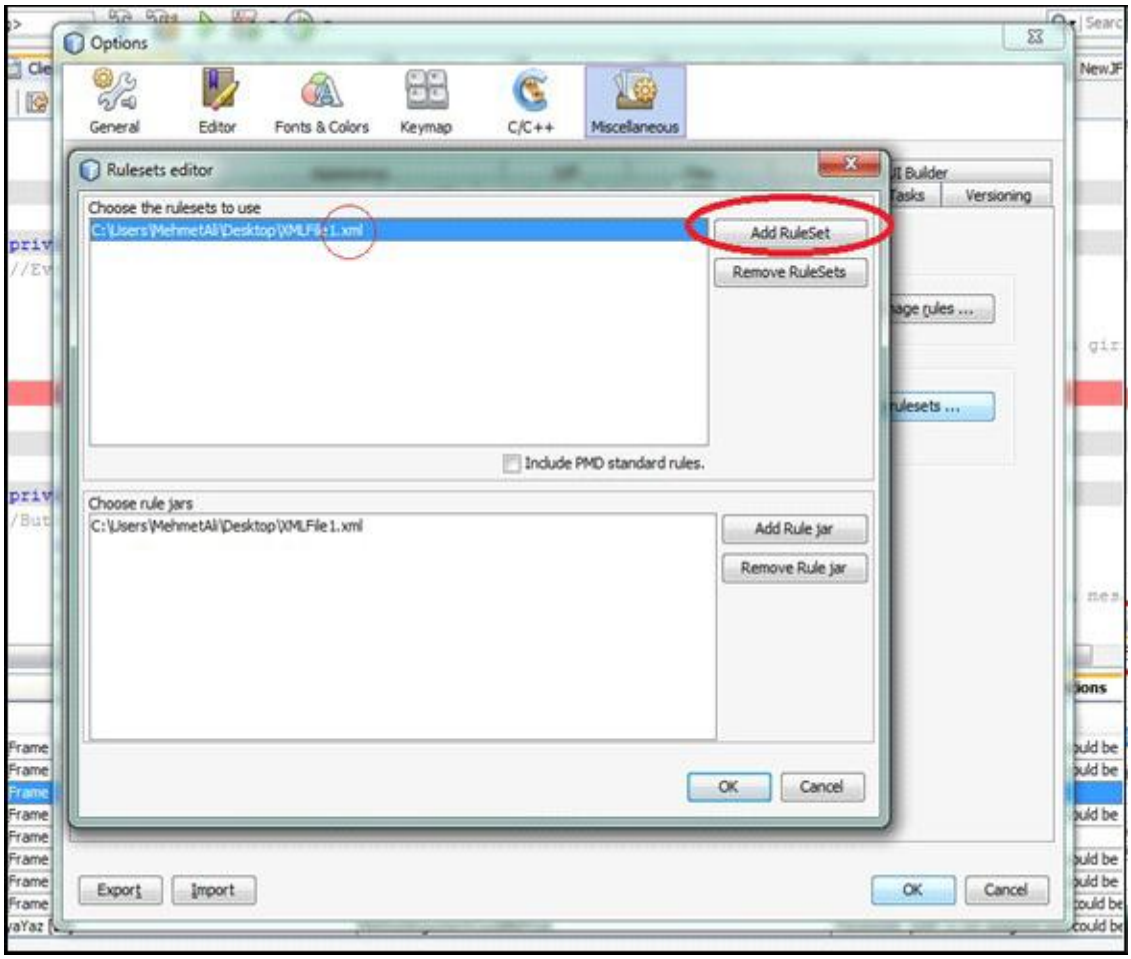


Figure 4.5 PMD Add Ruleset

When I started the test again with my custom ruleset downloaded from the internet, this time it gave me 26 warnings (see Figure 4.6).

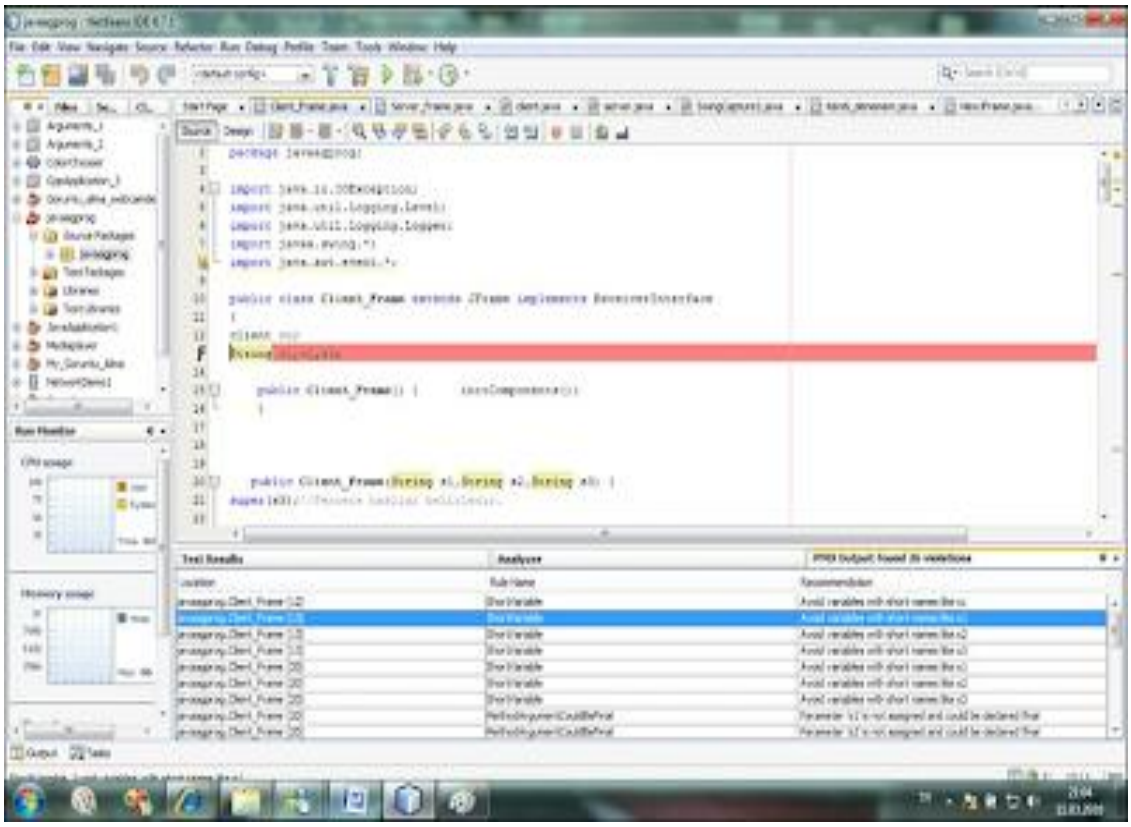


Figure 4.6 PMD Results with Custom Ruleset

4.1.2 Using PMD

It should be noted that not only Eclipse, but also a very useful add-on that can be added to many other IDEs (JDeveloper, JBuilder, IntelliJ IDEA etc.); PMD. The PMD extension stands out as Pretty Much Done, Project Mess Detector, Project Monitoring Directives, and so on. Your job is to scan the Java code and not just point to the code fragments that will cause a possible error, and suggest how to make them cleaner, smoother, more readable. If we were going to write a few of the draft rule sets roughly;

* Code blocks that can be expressed with better and / or less code

* Unused variables, parameters, methods

* Empty try / catch / finally / switch statements

* Complicated, difficult to read expressions, methods

* Unnecessary if clauses, for clauses that can be replaced with while

* Copy code blocks

* Errors that may occur due to copy / paste

* Variables that are too long or short named, methods

Of course, there is a full set of rules that evaluate the Java code outside of them. In practice, it is necessary to talk about the things that will be more confrontational.

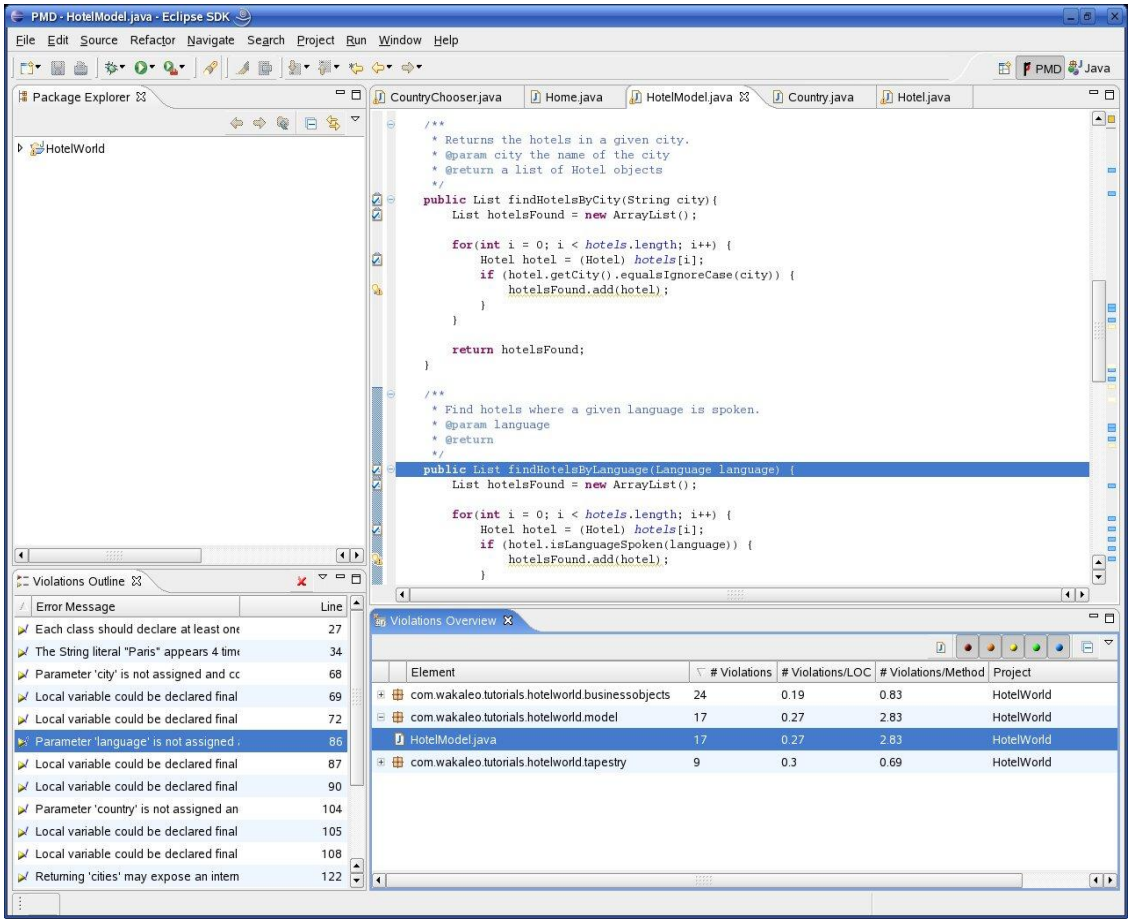


Figure 4.7 PMD Code Example

To give a few examples;

If you check the contents of the list variable returned by the getResultList () method of the Query class with list.isEmpty () instead of checking if (list.size ()> 0)

or String checking for equality in a variable as if (a.equals ("Ahmet")) as a friend, if (("Ah"). equals (a)) If you get rid of pointer errors, or another example of a BigDecimal type variable that we are assigning a value with a new BigDecimal ("0"), you might say that you do not use BigDecimal.ZERO, which is statically already in your possession. It is possible to duplicate such examples. (See figure 4.7)

We can edit the rule sets that the plugin has from the Window-> Preferences-> PMD menu. Thus, we can determine which rule assessment to take or how to evaluate it. Here sir, I long for the variable names to be understandable, you can interfere with my job, so you can tell me that you can only warn me about variables named longer than 20 characters.

4.2 Sonarqube

In this section, all the details of PMD tool and the usage of the tool is described.

4.2.1 Definition of Sonarqube

SonarQube (formerly Sonar) [24] is an open source platform developed by SonarSource for continuous inspection of code quality to perform automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages. SonarQube offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security vulnerabilities [25].

SonarQube can record metrics history and provides evolution graphs. SonarQube's provides fully automated analysis and integration with Maven, Ant, Gradle, MSBuild and continuous integration tools (Atlassian Bamboo, Jenkins, Hudson,etc.)[26].

SonarQube is a roof that collects other code analysis tools in one place. Almost all code analysis tools can be integrated. It is better to see it as a tool that facilitates the use of other tools and reporting [27].

Sonar comes with PMD, Checkstyle and Findbugs plugins in the first installation, and other plugins can be integrated afterwards.

We can define that the written codes comply with pre-determined quality criteria, without analyzing the codes.

- Finding bad codes,
- Identification of untested codes,
- Detection of repetitive code blocks,
- Capturing code that could lead to a security breach
- Reporting bad smelling codes

The end result of this analysis can be realized. Static code analysis can be automated with software tools. Visual analysis by people is called code review (code review). Static code analysis ensures that many errors can be found and corrected without having to go through the code, even if the code snippet does not hold its place.

4.2.2 Using Sonarqube

1. To scan a project, you first need to throw the sonar-project.properties file into the main directory of that project. I'm echoing a sample file. You need to update the following fields according to the procedure.

sonar.projectKey=Test

sonar.projectName=Test Project

sonar.projectVersion=1.0

Comma-separated paths to directories with sources (required)

#Java source home directory of files

sonar.sources=src/main/java

Main directory containing compiled classes

sonar.binaries=target/classes

2. Add the sonar runner bin directory to the "D: \ sonar \ sonar-runner-x.y \ bin \" environment variables.
3. From the command line, point to the directory where your progeny is located and run the "sonar-runner.bat" command.
4. You can see the results at http: // localhost: 9000 /. Here are the results of your progeny analysis. When you enter the detail, you can see the analysis according to the type of error according to severity.
5. When you enter the above address, you are logged in with a guest account. If you want to do detailed transactions, you have to login from the upper right corner. Username: admin, password: admin
6. You can operate on the profiles from the top "Quality Profiles" link on the top menu. From here, the scan rules can be processed and multiple profiles can be created.
7. To use Findbugs, we need to make the "Sonar way with Findbugs" profile the default profile on this page. Important Note: Since Findbugs needs the compiled code for analysis, your progeny should be compiled if you use this profile.

8. Again, we can process some rules from the same page and close some or add new rules.

5

Case Study

We did some tests with the open source java codes listed below. Codes were fetched from GitHub. Totally 979 lines of code from 12 java files have been analyzed with both PMD and SonarQube.

List of java files analyzed by the tools:

- CSVreader.java
- Distance.java
- EuclideanDistance.java
- Metric.java
- KMeans.java
- Kruskal.java
- Layer.java
- JavaApplication1.java
- Mlp.java
- MyFrame.java
- Neuron.java
- Server.java

5.1 PMD Configurations

The PMD command that we use for this experiment is given below:

pmd.bat -dir C:\code-analysis\sonar-codes -f html -rulesets java-basic,java-design,java-imports, java-unusedcode,java-coupling,java-comments,java-codesize,java-clone, java-controversial,java-empty,java-finalizers,java-naming,java-optimizations,java-sunsecure,java-strings,java-typeresolution,java-unnecessary, -encoding UTF-8 > "C:\code-analysis\all-pmd.html"

PMD checks the source code statically for the given rule sets and then using redirect operator “>” we forward the output to a file.

First parameter runs the pmd.bat file.

-dir parameter indicates the location of our source code to be analyzed

-f parameter is used for output format. In our case, it produces output in html format

-rulesets parameter is used for specifying the rules that pmd checks

5.2 SonarQube Configurations

The SonarQube server is first run using the file named StartSonart.bat. In the next step, we add following lines to “sonar-scanner.properties” file.

- sonar.projectKey=JavaAnalysisTool
- sonar.projectName=JavaAnalysisTool
- sonar.projectVersion=1.0
- sonar.sources=C:/code-analysis/sonar-codes
- sonar.language=java
- sonar.java.binaries=C:/code-analysis/sonar-codes

In order to start analysis, we execute the sonar-runner.bat command. Results can be displayed by accessing through http://localhost:9000 address.

5.3 SonarQube Results

After analyze, total of 145 issues were found by this tool. Complete analysis results are given in the Appendix. This tool provides variety of helpful information about detects found in the system or software to help developers fix the issues with the least possible cost and time.

There is a tab called issues which displays issues found of the projects and details such as type of the issues, severity level, effort time and etc. See figure 5.1.

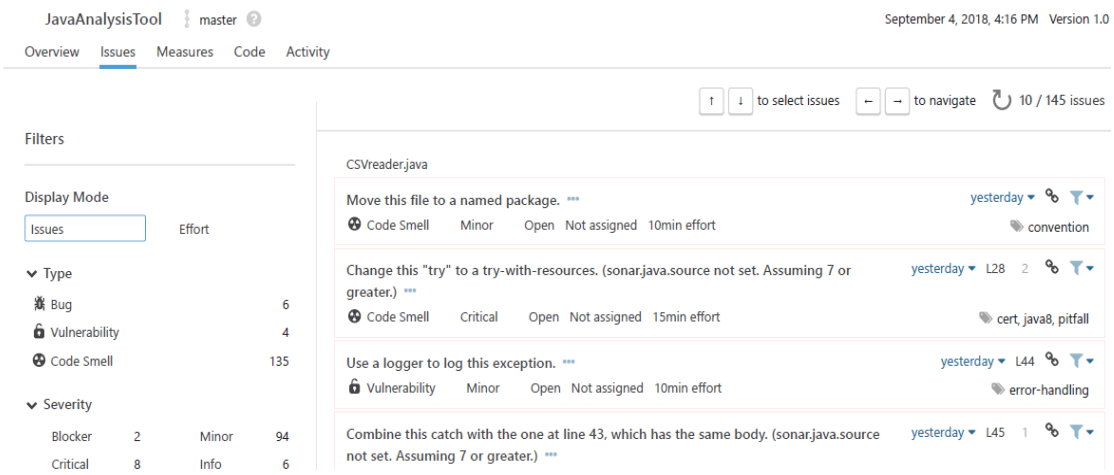


Figure 5.1 SonarQube Analyze result view

5.3.1 Type of the Issues

Three type of issues were detected by this tool which are bugs, code smell and vulnerability.

Software bugs can cause failure, error or fault in computer programs and systems and cause wrong and inappropriate results or even stop the program from running or behave as not requested. Total of 6 bugs were found after the test which were easy to solve due to the small quantity and help of the tool. A code smell is more complicated than software bugs. It can cause deeper problems in the systems and

software's. According to subject, developer and language can be decided what is code smell or it is not. 135 code smells were detected by the tool which should be considered by the developers to be corrected in order to run the program properly. Vulnerability problems are mostly related to security of the program or the system. It is a weakness or glitch that may cause damage. In our case study, 4 vulnerabilities were found. The results for the 12 analyzed files are shown in table 5.1.

Table 5.1 Type of issues found by SonarQube

Type	Number
Bug	6
Code smell	135
Vulnerability	4

5.3.2 Severity level of the Issues

One of the most important thing about the issues found by the tools is their severity level. As an example, one issue can cause to stop a software from running or one can just slow down the software speed. This tool can provide the severity level of the issues found to the developers and this can be a very handful information for them to deal with the issues. In addition, it will be very helpful to acknowledge the impact of the issue and the approximate time that they need to spend for this issue to be fixed. They can start from the bugs with the highest effect and prevent extra loss of time and cost. Some of the issues may be found even unnecessary to be fixed by the developers. This can happen because of a particular feature in the program which may occur infrequently. One of the other advantages of this future provided by the tool is that the developers can adjust each issues

according to their severity level. For example, less-experienced developers can deal with low severity level issues and much more experienced ones with high level problems.

The severity level of the found issues are divided in to 5 categories.

1. **BLOCKER**
   Bug with a high probability to impact the behavior of the application in production: memory leak, unclosed JDBC connection and etc. The code MUST be immediately fixed.
2. **CRITICAL**
   Either a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw: empty catch block, SQL injection and etc. The code MUST be immediately reviewed.
3. **MAJOR**
   Quality flaw which can highly impact the developer productivity: uncovered piece of code, duplicated blocks, unused parameters and etc.
4. **MINOR**
   Quality flaw which can slightly impact the developer productivity: lines should not be too long, "switch" statements should have at least 3 cases and etc.
5. **INFO**
   Neither a bug nor a quality flaw, just a finding.

This information can help developers to detect the most important bugs that must be fixed and find out the Project safety level according to the severity level of the issues found. See figure 5.2 for the results found.

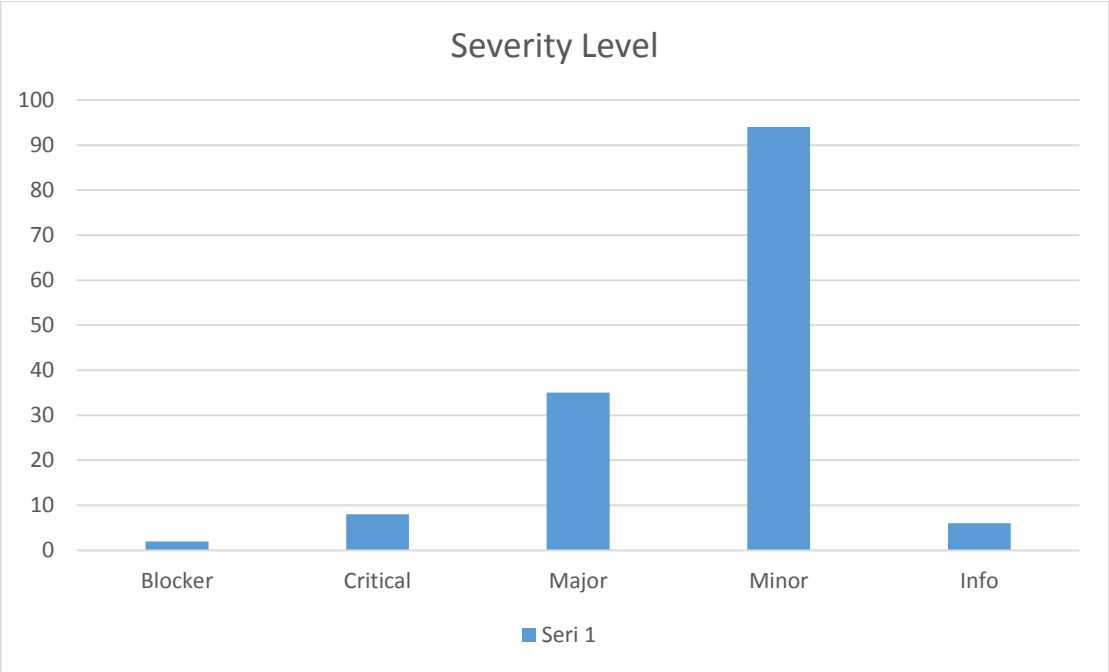


Figure 5.2 Severity levels found by SonarQube

5.3.3 Effort

The estimated time that the developer needs to fix the issue is presented as effort time in SonarQube. The results are shown in figure 5.3. %30 of the issues estimated effort time is 10 minutes. Almost %50 of the issues effort time is 5 min or less. The effort time for 6 issues are not specified.

In order to test this time and see how accurate the results are, we choose few issues from each category and calculate the time that it takes to fix them. It turns out that if sonar estimates 5 minutes, it may take much longer because after the issue has been fixed, we need to test it if it works. For some of the issues, after the test failed, there was also extra time to fix it again so the estimate time was not accurate for these type of examples.

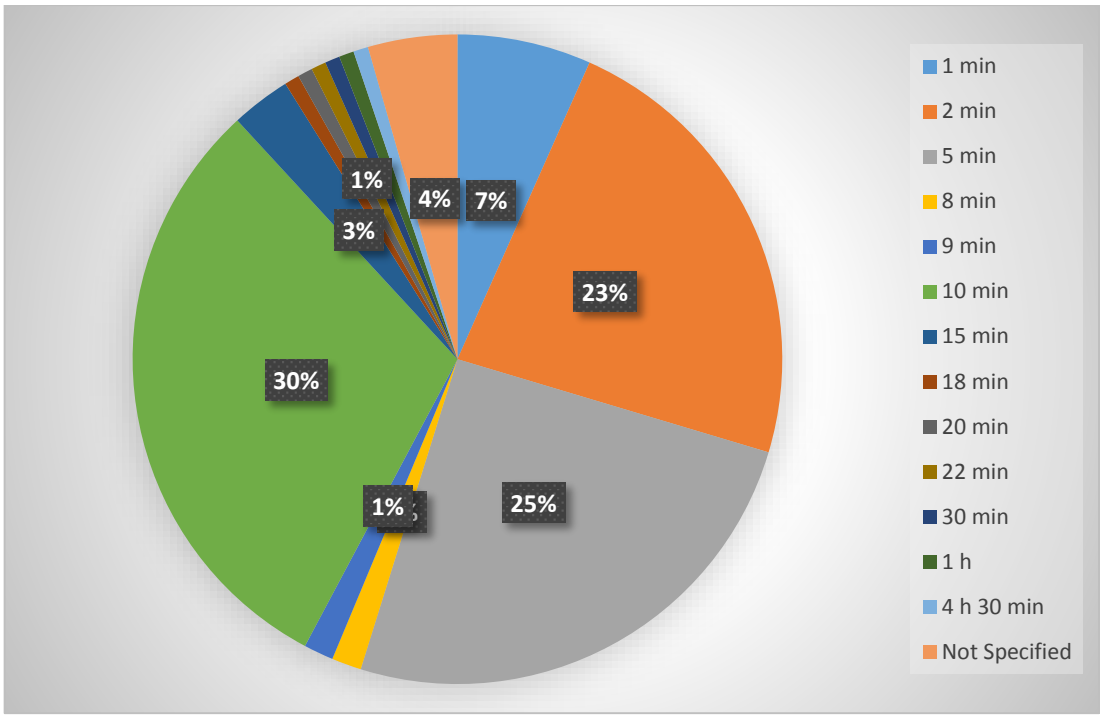


Figure 5.3 Effort time of the issues found by SonarQube

Between all the results, there is one issue with effort time 4 hour and 30 min which caught our attention to go deeply and check what that is. One of the classes has 6 parents. Each class is authorized to have maximum 5 parents so it is a major issue. In order to correct this issue almost whole codes in the class should be changed and lower the parents class to 5 which takes even more than the estimated effort time to fix the issue.

5.4 PMD Results

The output file has total of 701 problems listed. After checking the list we realized that there are many duplicate problems stated for the same line of the codes. The results of PMD are shown very simple in the file as you can see in the figure 5.4 below. The file, line and the problem are stated in the list.

PMD report

Problems found

#	File	Line	Problem
1	C:\code-analysis\sonar-codes\CSVreader.java	12	All classes and interfaces must belong to a named package
2	C:\code-analysis\sonar-codes\CSVreader.java	12	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
3	C:\code-analysis\sonar-codes\CSVreader.java	12	The class 'CSVreader' has a Modified Cyclomatic Complexity of 8 (Highest = 10).
4	C:\code-analysis\sonar-codes\CSVreader.java	12	The class 'CSVreader' has a Standard Cyclomatic Complexity of 8 (Highest = 10).
5	C:\code-analysis\sonar-codes\CSVreader.java	17	Avoid prefixing parameters by in, out or inOut. Uses Javadoc to document this behavior.
6	C:\code-analysis\sonar-codes\CSVreader.java	17	Parameter 'columns' is not assigned and could be declared final
7	C:\code-analysis\sonar-codes\CSVreader.java	17	Parameter 'inFile' is not assigned and could be declared final
8	C:\code-analysis\sonar-codes\CSVreader.java	17	Parameter 'rows' is not assigned and could be declared final
9	C:\code-analysis\sonar-codes\CSVreader.java	17	The method 'read' has a Modified Cyclomatic Complexity of 10.
10	C:\code-analysis\sonar-codes\CSVreader.java	17	The method 'read' has a Standard Cyclomatic Complexity of 10.
11	C:\code-analysis\sonar-codes\CSVreader.java	17	The method 'read(String, int, int)' has a cyclomatic complexity of 14.
12	C:\code-analysis\sonar-codes\CSVreader.java	21	Avoid variables with short names like bf
13	C:\code-analysis\sonar-codes\CSVreader.java	21	Found 'DD'-anomaly for variable 'bf' (lines '21'-'29').
14	C:\code-analysis\sonar-codes\CSVreader.java	22	Found 'DD'-anomaly for variable 'arr' (lines '22'-'38').
15	C:\code-analysis\sonar-codes\CSVreader.java	22	Found 'DU'-anomaly for variable 'arr' (lines '22'-'61').
16	C:\code-analysis\sonar-codes\CSVreader.java	24	Found 'DD'-anomaly for variable 'line' (lines '24'-'31').
17	C:\code-analysis\sonar-codes\CSVreader.java	25	Avoid variables with short names like r
18	C:\code-analysis\sonar-codes\CSVreader.java	25	Found 'DD'-anomaly for variable 'r' (lines '25'-'40').

Figure 5.4 PMD Report

The file column, is giving the source file details which the problem has been found. The line column, is simply pointing out the line of the code to help developers find the problem faster and easier and finally the problem column is giving a summary of the problem found on that line.

At first sight, it looks like the PMD is not giving any other specification about the problems found but after examining the problems one by one, there are many other details given for each problem. On the problem column, after clicking on the problem another page is being opened. On that page the other details are provided to the developers which will be an asset to help them solve the problems faster. See figure 5.5.

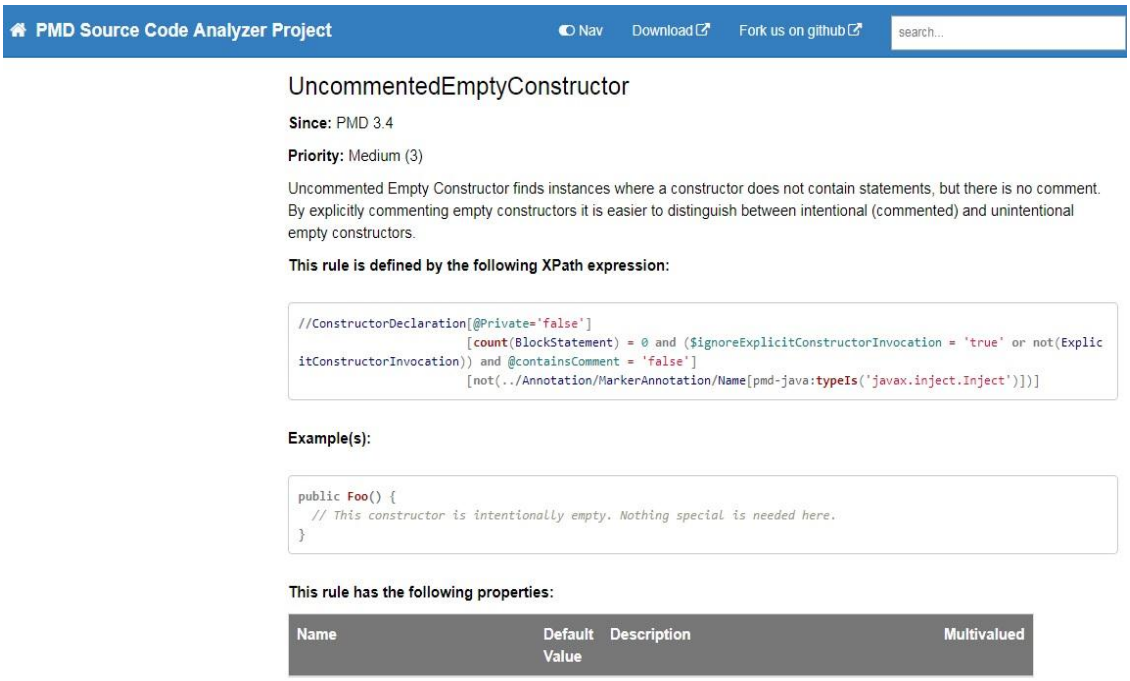


Figure 5.5 PMD Report detailed about the rules and issues

5.4.1 Priority

On the problem column, after clicking on the problem, another page will be opened which is providing extra details about it. One of the important parts is the priority of that issue. In PMD tool the priorities are listed as one of the below categories:

- Low
- Medium
- Medium-low
- Medium-high
- High

In these categories, the priority is defined as a digit from 1 to 5 which the 5 is the lowest and 1 is the highest level.

5.4.2 Other results

Another details is the version of the PMD which this rule is available from. But in some of the rules the word 'Deprecated' is written on top of the version which means this rule will be removed and replaced by another rule.

In addition, there is a small description about that rule and the Java class which the rule is defined by and followed by the description an example is provided to help the developers.

Also, the properties of the rule is written on a table. See figure 5.6.

This rule has the following properties:

Name	Default Value	Description	Multivalued
classReportLevel	80	Total class complexity reporting threshold	no
methodReportLevel	10	Cyclomatic complexity reporting threshold	no
cycloOptions		Choose options for the computation of Cyclo	yes. Delimiter is '|'.
reportLevel	10	Deprecated Cyclomatic Complexity reporting threshold	no

Figure 5.6 Rule Properties

In the table details like name, default value, description and multivalued are provided.

As an example, in the above table the reportLevel property is deprecated and replaced by another now in earlier versions of PMD.

The final useful information is the rule shared with the developer so it can be used easily. The developers can customize the properties and their value according to their need which is one of the most advantages of PMD as we mentioned earlier while introducing this useful tool.

5.5 Comparison of PMD and SonarQube

Results are very similar for the 12 java files. For the sake of demonstration we present comparison for CSVreader.java, Distance.java and Server.java in table 5.2.

Table 5.2 Comparison of CSVreader.java

SonarQube	**PMD**
Move this file to a named package.	All classes and interfaces must belong to a named package
Change this "try" to a try-with-resources. (sonar.java.source not set. Assuming 7 or greater.	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
Use a logger to log this exception.	Avoid prefixing parameters by in, out or inOut. Uses Javadoc to document this behavior.
Combine this catch with the one at line 43, which has the same body. (sonar.java.source not set. Assuming 7 or greater.)	Parameter 'columns' is not assigned and could be declared final
Replace this use of System.out or System.err by a logger.	Avoid variables with short names like bf
	Found 'DD'-anomaly for variable 'bf'
	Avoid assignments in operands
	Parameter 'args' is not assigned and could be declared final
	Avoid using Literals in Conditional Statements

Table 5.3 Comparison of Distance.java

SonarQube	PMD
Move this file to a named package.	Comment is too large: Too many lines
	All classes and interfaces must belong to a named package
	Avoid using short method names
	Unnecessary modifier 'public' on method 'd': the method is declared in an interface type
	Avoid variables with short names like x

Table 5.4 Comparison of Server.java

SonarQube	PMD
Move this file to a named package.	All classes and interfaces must belong to a named package
Use classes from the Java API instead of Sun classes.	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
Refactor your code to get this URI from a customizable parameter.	Public method and constructor comments are required

Table 5.4 Comparison of Server.java (Continued)

Add logic to this catch clause or eliminate it and rethrow the exception automatically.	Local variable 'port' could be declared final
Replace this use of System.out or System.err by a logger.	Avoid variables with short names like t
	Found 'DU'-anomaly for variable 'read' (lines '44'-'51').
	Avoid assignments in operands
	Parameter t is not assigned and could be declared final
	To avoid mistakes add a comment at the beginning of the readContent method if you want a default access modifier
	Potential violation of Law of Demeter (method chain calls)
	Header comments are required
	The utility class name 'Server' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'
	A method/constructor should not explicitly throw java.lang.Exception
	Use explicit scoping instead of the default package private level

5.6 Tools in contrast with review

After analyzing the results of both tools and how they help the developers in software testing, as final step, an informal review was done for one of the projects called 'Layer' to find answers for the questions below [34]:

- Are there any other bugs that are not found by the tools?
- What type of bugs were not found by the tools?
- How accurate are the bugs found by the tools?
- Which tools result is closer to the review?
- Can the developers rely on the tools test only?

The codes were inspected very carefully in a review meeting. Then the results of both tools were compared with the review results to check if they were found by the review also.

5.6.1 SonarQube vs review

The SonarQube results for this specific file were compared with the results of review to see if they have been all found by the review or not. See table 5.5.

After analyzing the results carefully, there are some defects that were not found by the review.

These are mostly related to the programming language standards which are related to names in our case study. As an example, SonarQube found 5 local variables in the file which their names do not match the regular expressions, however these defects were not recognized by the review.

Using tools for these type of defects are the most appropriate approach since they can do it so much faster and it will cost less. Developers mostly do not recognize them because it is not easy to check them all one by one and they are very time consuming.

Table 5.5 SonarQube results compared to the review

Defect type	SonarQube	Review
Move this file to a named package.	✓	✓
Rename this local variable to match the regular expression'^ [a-z][a-zA-Z0-9]*$'.	✓	x
Replace the type specification in this constructor call with the diamond operator ("<>").	✓	✓
Rename this method name to match the regular expression '^ [a-z][a-zA-Z0-9]*$'.	✓	x
Move the array designator from the variable to the type.	✓	✓
Declare "_prev_n_neurons" on a separate line.	✓	✓
Rename this field "_n_neurons" to match the regular expression '^ [a-z][a-zA-Z0-9]*$'.	✓	x
Rename this field "_prev_n_neurons" to match the regular expression '^ [a-z][a-zA-Z0-9]*$'.	✓	x
Move this file to a named package.	✓	✓
Remove this unused method parameter "evt".	✓	✓
Replace this use of System.out or System.err by a logger.	✓	✓

5.6.2 PMD vs Review

The PMD results compared to the review has almost the same conclusions like the SonarQube. See table 5.6.

Table 5.6 PMD results compared to the review

Defect Type	PMD	Review
All classes and interfaces must belong to a named package	✓	✓
Only variables that are final should contain underscores, 'n_neurons' is not final.	✓	x
Parameter 'n_neurons' is not assigned and could be declared final	✓	✓
Parameter 'prev_n_neurons' is not assigned	✓	✓
Avoid instantiating new objects inside loops	✓	x
Avoid variables with short names like in	✓	x
Found 'DD'-anomaly for variable 'out' (lines '24'-'26').	✓	✓
Avoid if (x != y) ..; else ..;	✓	✓
Useless parentheses.	✓	x
The static method name 'add_bias' doesn't match '[a-z][a-zA-Z0-9]*'	✓	x
Public method and constructor comments are required	✓	✓
Potential violation of Law of Demeter (method chain calls)	✓	✓
Avoid prefixing parameters by in, out or inOut.	✓	✓

The same type of defects which were not found by the review were found by PMD. Most of them are related to programming standards and are not major issues.

After checking both tools with the review, there were some defects that were not found by the tools. They are listed in table 5.7. Most of these defects are logical type and were detected by following the code and using test cases by different input values.

In summary, for logical type of defects the tools were not as successful as the review however for the common type of issues, it is better to use tools since this process will be much cheaper and faster. Developers cannot rely on these tools %100 to solve all the defects of the software. Tools needs some improvements for better results especially for logical type of issues. Some of the below defects can even cause the software to crash which can have bad consequences so it is very important to add some new features to the tools to improve them.

Table 5.7 Defect results found in review

Defect types	Occurrences
Data Range Inappropriate and not checked	2
Inappropriate result	3
Incomplete comparison	2
Wrong Input	1
Length may be less than zero	1
Other logical errors	5

6

RESULTS AND DISCUSSIONS

Even though PMD analysis report seems to be more detailed, actually SonarQube provides more meaningful and deep expressions. For instance, SonarQube suggests followings:

- Refactor your code to get this URI from a customizable parameter.
- Add logic to this catch clause or eliminate it and rethrow the exception automatically.

PMD could not suggest such changes above. Instead, PMD generally suggests not using short variable, using comments and all other syntax rules. It identifies potential problems mainly dead and duplicated code, cyclomatic complexity and overcomplicated expressions. Important thing about PMD is that It analyses Abstract Syntax Tree(AST) generated by JavaCC and does not require actual compilation. Thus no need to provide .class files to PMD. In contrast to PMD, SonarQube requires compiled .class files. PMD only does analysis statically whereas SonarQube does static and dynamic analysis. What makes SonarQube really stand out is that it not only provides metrics and statistics about our code, but translates these non-descript values to real business values such as risk and technical debt. SonarQube not only addresses core developers and programmers but, project managers and even higher managerial levels due to the management aspect it offers. This concept is further strengthened by SonarQube's enhanced reporting capabilities and multiple views addressing source code from different perspective.

In addition, the logical type of defects were only found by the review so the tools needs to be developed to be able to find or at least identify these type of defects. Using tools can save time and cost while being used by other methods of testing.

A

PMD Code Results

#	File	Line	Problem
1	C:\code-analysis\sonar-codes\CSVreader.java	12	All classes and interfaces must belong to a named package
2	C:\code-analysis\sonar-codes\CSVreader.java	12	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
3	C:\code-analysis\sonar-codes\CSVreader.java	12	The class 'CSVreader' has a Modified Cyclomatic Complexity of 8 (Highest = 10).
4	C:\code-analysis\sonar-codes\CSVreader.java	12	The class 'CSVreader' has a Standard Cyclomatic Complexity of 8 (Highest = 10).
5	C:\code-analysis\sonar-codes\CSVreader.java	17	Avoid prefixing parameters by in, out or inOut. Uses Javadoc to document this behavior.
6	C:\code-analysis\sonar-codes\CSVreader.java	17	Parameter 'columns' is not assigned and could be declared final
7	C:\code-analysis\sonar-codes\CSVreader.java	17	Parameter 'inFile' is not assigned and could be declared final
8	C:\code-analysis\sonar-codes\CSVreader.java	17	Parameter 'rows' is not assigned and could be declared final

9	C:\code-analysis\sonar-codes\CSVreader.java	17	The method 'read' has a Modified Cyclomatic Complexity of 10.
10	C:\code-analysis\sonar-codes\CSVreader.java	17	The method 'read' has a Standard Cyclomatic Complexity of 10.
11	C:\code-analysis\sonar-codes\CSVreader.java	17	The method 'read(String, int, int)' has a cyclomatic complexity of 14.
12	C:\code-analysis\sonar-codes\CSVreader.java	21	Avoid variables with short names like bf
13	C:\code-analysis\sonar-codes\CSVreader.java	21	Found 'DD'-anomaly for variable 'bf' (lines '21'-'29').
14	C:\code-analysis\sonar-codes\CSVreader.java	22	Found 'DD'-anomaly for variable 'arr' (lines '22'-'38').
15	C:\code-analysis\sonar-codes\CSVreader.java	22	Found 'DU'-anomaly for variable 'arr' (lines '22'-'61').
16	C:\code-analysis\sonar-codes\CSVreader.java	24	Found 'DD'-anomaly for variable 'line' (lines '24'-'31').
17	C:\code-analysis\sonar-codes\CSVreader.java	25	Avoid variables with short names like r
18	C:\code-analysis\sonar-codes\CSVreader.java	25	Found 'DD'-anomaly for variable 'r' (lines '25'-'40').
19	C:\code-analysis\sonar-codes\CSVreader.java	25	Found 'DU'-anomaly for variable 'r' (lines '25'-'61').

20	C:\code-analysis\sonar-codes\CSVreader.java	26	Avoid variables with short names like c
21	C:\code-analysis\sonar-codes\CSVreader.java	31	Avoid assignments in operands
22	C:\code-analysis\sonar-codes\CSVreader.java	31	Found 'DU'-anomaly for variable 'line' (lines '31'-'61').
23	C:\code-analysis\sonar-codes\CSVreader.java	32	Avoid variables with short names like x
24	C:\code-analysis\sonar-codes\CSVreader.java	32	Local variable 'x' could be declared final
25	C:\code-analysis\sonar-codes\CSVreader.java	38	Found 'DD'-anomaly for variable 'arr' (lines '38'-'38').
26	C:\code-analysis\sonar-codes\CSVreader.java	38	Found 'DU'-anomaly for variable 'arr' (lines '38'-'61').
27	C:\code-analysis\sonar-codes\CSVreader.java	38	Potential violation of Law of Demeter (method chain calls)
28	C:\code-analysis\sonar-codes\CSVreader.java	40	Found 'DD'-anomaly for variable 'r' (lines '40'-'40').
29	C:\code-analysis\sonar-codes\CSVreader.java	40	Found 'DU'-anomaly for variable 'r' (lines '40'-'61').
30	C:\code-analysis\sonar-codes\CSVreader.java	63	Parameter 'args' is not assigned and could be declared final

31	C:\code-analysis\sonar-codes\CSVreader.java	63	Public method and constructor comments are required
32	C:\code-analysis\sonar-codes\CSVreader.java	67	Found 'DD'-anomaly for variable 'print' (lines '67'-'74').
33	C:\code-analysis\sonar-codes\CSVreader.java	67	Found 'DD'-anomaly for variable 'print' (lines '67'-'82').
34	C:\code-analysis\sonar-codes\CSVreader.java	70	Avoid using Literals in Conditional Statements
35	C:\code-analysis\sonar-codes\CSVreader.java	72	Potential violation of Law of Demeter (method chain calls)
36	C:\code-analysis\sonar-codes\CSVreader.java	73	Potential violation of Law of Demeter (method chain calls)
37	C:\code-analysis\sonar-codes\CSVreader.java	74	Potential violation of Law of Demeter (method chain calls)
38	C:\code-analysis\sonar-codes\CSVreader.java	85	Found 'DU'-anomaly for variable 'test' (lines '85'-'96').
39	C:\code-analysis\sonar-codes\CSVreader.java	85	Local variable 'test' could be declared final
40	C:\code-analysis\sonar-codes\CSVreader.java	90	Potential violation of Law of Demeter (method chain calls)
41	C:\code-analysis\sonar-codes\CSVreader.java	90	Potential violation of Law of Demeter (method chain calls)

42	C:\code-analysis\sonar-codes\Distance.java	1	Comment is too large: Too many lines
43	C:\code-analysis\sonar-codes\Distance.java	19	Comment is too large: Too many lines
44	C:\code-analysis\sonar-codes\Distance.java	32	All classes and interfaces must belong to a named package
45	C:\code-analysis\sonar-codes\Distance.java	36	Avoid using short method names
46	C:\code-analysis\sonar-codes\Distance.java	36	Avoid variables with short names like x
47	C:\code-analysis\sonar-codes\Distance.java	36	Avoid variables with short names like y
48	C:\code-analysis\sonar-codes\Distance.java	36	Unnecessary modifier 'public' on method 'd': the method is declared in an interface type
49	C:\code-analysis\sonar-codes\Distance.java	36	Unnecessary modifier 'public' on method 'd': the method is declared in an interface type
50	C:\code-analysis\sonar-codes\EuclideanDistance.java	1	Comment is too large: Too many lines
51	C:\code-analysis\sonar-codes\EuclideanDistance.java	21	Comment is too large: Too many lines
52	C:\code-analysis\sonar-codes\EuclideanDistance.java	30	All classes and interfaces must belong to a named package

53	C:\code-analysis\sonar-codes\EuclideanDistance.java	36	Avoid using redundant field initializer for 'weight'
54	C:\code-analysis\sonar-codes\EuclideanDistance.java	41	Document empty constructor
55	C:\code-analysis\sonar-codes\EuclideanDistance.java	49	Consider using varargs for methods or constructors which take an array the last parameter.
56	C:\code-analysis\sonar-codes\EuclideanDistance.java	49	Parameter 'weight' is not assigned and could be declared final
57	C:\code-analysis\sonar-codes\EuclideanDistance.java	49	The user-supplied array 'weight' is stored directly.
58	C:\code-analysis\sonar-codes\EuclideanDistance.java	60	Avoid if (x != y) ..; else ..;
59	C:\code-analysis\sonar-codes\EuclideanDistance.java	61	A method should have only one exit point, and that should be the last statement in the method
60	C:\code-analysis\sonar-codes\EuclideanDistance.java	70	Avoid using short method names
61	C:\code-analysis\sonar-codes\EuclideanDistance.java	70	Avoid variables with short names like x
62	C:\code-analysis\sonar-codes\EuclideanDistance.java	70	Avoid variables with short names like y
63	C:\code-analysis\sonar-codes\EuclideanDistance.java	70	Consider using varargs for methods or constructors which take an array the last parameter.

64	C:\code-analysis\sonar-codes\EuclideanDistance.java	70	Parameter 'x' is not assigned and could be declared final
65	C:\code-analysis\sonar-codes\EuclideanDistance.java	70	Parameter 'y' is not assigned and could be declared final
66	C:\code-analysis\sonar-codes\EuclideanDistance.java	74	Found 'DU'-anomaly for variable 'dist' (lines '74'-'92').
67	C:\code-analysis\sonar-codes\EuclideanDistance.java	78	Avoid variables with short names like d
68	C:\code-analysis\sonar-codes\EuclideanDistance.java	78	Local variable 'd' could be declared final
69	C:\code-analysis\sonar-codes\EuclideanDistance.java	86	Avoid variables with short names like d
70	C:\code-analysis\sonar-codes\EuclideanDistance.java	86	Local variable 'd' could be declared final
71	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	Avoid using short method names
72	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	Avoid variables with short names like x
73	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	Avoid variables with short names like y
74	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	Consider using varargs for methods or constructors which take an array the last parameter.

75	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	Parameter 'x' is not assigned and could be declared final
76	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	Parameter 'y' is not assigned and could be declared final
77	C:\code-analysis\sonar-codes\EuclideanDistance.java	101	The method 'd(float, float)' has a cyclomatic complexity of 13.
78	C:\code-analysis\sonar-codes\EuclideanDistance.java	105	Avoid variables with short names like n
79	C:\code-analysis\sonar-codes\EuclideanDistance.java	105	Found 'DU'-anomaly for variable 'n' (lines '105'-'137').
80	C:\code-analysis\sonar-codes\EuclideanDistance.java	105	Local variable 'n' could be declared final
81	C:\code-analysis\sonar-codes\EuclideanDistance.java	106	Avoid variables with short names like m
82	C:\code-analysis\sonar-codes\EuclideanDistance.java	106	Found 'DD'-anomaly for variable 'm' (lines '106'-'112').
83	C:\code-analysis\sonar-codes\EuclideanDistance.java	106	Found 'DD'-anomaly for variable 'm' (lines '106'-'123').
84	C:\code-analysis\sonar-codes\EuclideanDistance.java	106	Found 'DU'-anomaly for variable 'm' (lines '106'-'137').
85	C:\code-analysis\sonar-codes\EuclideanDistance.java	107	Found 'DD'-anomaly for variable 'dist' (lines '107'-'131').

86	C:\code-analysis\sonar-codes\EuclideanDistance.java	107	Found 'DD'-anomaly for variable 'dist' (lines '107'-'133').
87	C:\code-analysis\sonar-codes\EuclideanDistance.java	107	Found 'DU'-anomaly for variable 'dist' (lines '107'-'137').
88	C:\code-analysis\sonar-codes\EuclideanDistance.java	111	Potential violation of Law of Demeter (method chain calls)
89	C:\code-analysis\sonar-codes\EuclideanDistance.java	111	Potential violation of Law of Demeter (method chain calls)
90	C:\code-analysis\sonar-codes\EuclideanDistance.java	112	Found 'DD'-anomaly for variable 'm' (lines '112'-'112').
91	C:\code-analysis\sonar-codes\EuclideanDistance.java	113	Avoid variables with short names like d
92	C:\code-analysis\sonar-codes\EuclideanDistance.java	113	Local variable 'd' could be declared final
93	C:\code-analysis\sonar-codes\EuclideanDistance.java	122	Potential violation of Law of Demeter (method chain calls)
94	C:\code-analysis\sonar-codes\EuclideanDistance.java	122	Potential violation of Law of Demeter (method chain calls)
95	C:\code-analysis\sonar-codes\EuclideanDistance.java	123	Found 'DD'-anomaly for variable 'm' (lines '123'-'123').
96	C:\code-analysis\sonar-codes\EuclideanDistance.java	124	Avoid variables with short names like d

97	C:\code-analysis\sonar-codes\EuclideanDistance.java	124	Local variable 'd' could be declared final
98	C:\code-analysis\sonar-codes\EuclideanDistance.java	147	Avoid using short method names
99	C:\code-analysis\sonar-codes\EuclideanDistance.java	147	Avoid variables with short names like x
100	C:\code-analysis\sonar-codes\EuclideanDistance.java	147	Avoid variables with short names like y
101	C:\code-analysis\sonar-codes\EuclideanDistance.java	147	Parameter 'x' is not assigned and could be declared final
102	C:\code-analysis\sonar-codes\EuclideanDistance.java	147	Parameter 'y' is not assigned and could be declared final
103	C:\code-analysis\sonar-codes\EuclideanDistance.java	147	The method 'd(double, double)' has a cyclomatic complexity of 13.
104	C:\code-analysis\sonar-codes\EuclideanDistance.java	151	Avoid variables with short names like n
105	C:\code-analysis\sonar-codes\EuclideanDistance.java	151	Found 'DU'-anomaly for variable 'n' (lines '151'-'182').
106	C:\code-analysis\sonar-codes\EuclideanDistance.java	151	Local variable 'n' could be declared final
107	C:\code-analysis\sonar-codes\EuclideanDistance.java	152	Avoid variables with short names like m

108	C:\code-analysis\sonar-codes\EuclideanDistance.java	152	Found 'DD'-anomaly for variable 'm' (lines '152'-'158').
109	C:\code-analysis\sonar-codes\EuclideanDistance.java	152	Found 'DD'-anomaly for variable 'm' (lines '152'-'169').
110	C:\code-analysis\sonar-codes\EuclideanDistance.java	152	Found 'DU'-anomaly for variable 'm' (lines '152'-'182').
111	C:\code-analysis\sonar-codes\EuclideanDistance.java	153	Found 'DD'-anomaly for variable 'dist' (lines '153'-'177').
112	C:\code-analysis\sonar-codes\EuclideanDistance.java	153	Found 'DD'-anomaly for variable 'dist' (lines '153'-'179').
113	C:\code-analysis\sonar-codes\EuclideanDistance.java	153	Found 'DU'-anomaly for variable 'dist' (lines '153'-'182').
114	C:\code-analysis\sonar-codes\EuclideanDistance.java	157	Potential violation of Law of Demeter (method chain calls)
115	C:\code-analysis\sonar-codes\EuclideanDistance.java	157	Potential violation of Law of Demeter (method chain calls)
116	C:\code-analysis\sonar-codes\EuclideanDistance.java	158	Found 'DD'-anomaly for variable 'm' (lines '158'-'158').
117	C:\code-analysis\sonar-codes\EuclideanDistance.java	159	Avoid variables with short names like d
118	C:\code-analysis\sonar-codes\EuclideanDistance.java	159	Local variable 'd' could be declared final

119	C:\code-analysis\sonar-codes\EuclideanDistance.java	168	Potential violation of Law of Demeter (method chain calls)
120	C:\code-analysis\sonar-codes\EuclideanDistance.java	168	Potential violation of Law of Demeter (method chain calls)
121	C:\code-analysis\sonar-codes\EuclideanDistance.java	169	Found 'DD'-anomaly for variable 'm' (lines '169'-'169').
122	C:\code-analysis\sonar-codes\EuclideanDistance.java	170	Avoid variables with short names like d
123	C:\code-analysis\sonar-codes\EuclideanDistance.java	170	Local variable 'd' could be declared final
124	C:\code-analysis\sonar-codes\JavaApplication1.java	10	All classes and interfaces must belong to a named package
125	C:\code-analysis\sonar-codes\JavaApplication1.java	10	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
126	C:\code-analysis\sonar-codes\JavaApplication1.java	12	Public method and constructor comments are required
127	C:\code-analysis\sonar-codes\JavaApplication1.java	12	The static method name 'func_file' doesn't match '[a-z][a-zA-Z0-9]*'
128	C:\code-analysis\sonar-codes\JavaApplication1.java	13	Local variable 'the_file' could be declared final

129	C:\code-analysis\sonar-codes\JavaApplication1.java	13	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'the_file' is not final.
130	C:\code-analysis\sonar-codes\JavaApplication1.java	21	Local variable 'the_output' could be declared final
131	C:\code-analysis\sonar-codes\JavaApplication1.java	21	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'the_output' is not final.
132	C:\code-analysis\sonar-codes\JavaApplication1.java	34	Local variable 'the_input' could be declared final
133	C:\code-analysis\sonar-codes\JavaApplication1.java	34	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'the_input' is not final.
134	C:\code-analysis\sonar-codes\JavaApplication1.java	36	Found 'DD'-anomaly for variable 'the_line' (lines '36'-'39').
135	C:\code-analysis\sonar-codes\JavaApplication1.java	36	Found 'DU'-anomaly for variable 'the_line' (lines '36'-'47').
136	C:\code-analysis\sonar-codes\JavaApplication1.java	36	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'the_line' is not final.
137	C:\code-analysis\sonar-codes\JavaApplication1.java	49	Parameter 'args' is not assigned and could be declared final
138	C:\code-analysis\sonar-codes\JavaApplication1.java	49	Public method and constructor comments are required
139	C:\code-analysis\sonar-codes\KMeans.java	1	Comment is too large: Too many lines

140	C:\code-analysis\sonar-codes\KMeans.java	25	All classes and interfaces must belong to a named package
141	C:\code-analysis\sonar-codes\KMeans.java	25	The class 'KMeans' has a Modified Cyclomatic Complexity of 3 (Highest = 14).
142	C:\code-analysis\sonar-codes\KMeans.java	25	The class 'KMeans' has a Standard Cyclomatic Complexity of 3 (Highest = 14).
143	C:\code-analysis\sonar-codes\KMeans.java	25	This class has too many methods, consider refactoring it.
144	C:\code-analysis\sonar-codes\KMeans.java	31	Avoid variables with short names like k
145	C:\code-analysis\sonar-codes\KMeans.java	32	Field comments are required
146	C:\code-analysis\sonar-codes\KMeans.java	35	Field comments are required
147	C:\code-analysis\sonar-codes\KMeans.java	36	Avoid variables with short names like pp
148	C:\code-analysis\sonar-codes\KMeans.java	36	Field comments are required
149	C:\code-analysis\sonar-codes\KMeans.java	37	Field comments are required
150	C:\code-analysis\sonar-codes\KMeans.java	38	Field comments are required

151	C:\code-analysis\sonar-codes\KMeans.java	40	Field comments are required
152	C:\code-analysis\sonar-codes\KMeans.java	40	Variables should start with a lowercase character, 'L1norm' starts with uppercase character.
153	C:\code-analysis\sonar-codes\KMeans.java	43	Avoid variables with short names like m
154	C:\code-analysis\sonar-codes\KMeans.java	43	Field comments are required
155	C:\code-analysis\sonar-codes\KMeans.java	44	Avoid variables with short names like n
156	C:\code-analysis\sonar-codes\KMeans.java	44	Field comments are required
157	C:\code-analysis\sonar-codes\KMeans.java	47	Comment is too large: Line too long
158	C:\code-analysis\sonar-codes\KMeans.java	47	Field comments are required
159	C:\code-analysis\sonar-codes\KMeans.java	48	Field comments are required
160	C:\code-analysis\sonar-codes\KMeans.java	49	Field comments are required
161	C:\code-analysis\sonar-codes\KMeans.java	49	The field name indicates a constant but its modifiers do not

162	C:\code-analysis\sonar-codes\KMeans.java	49	Variables should start with a lowercase character, 'WCSS' starts with uppercase character.
163	C:\code-analysis\sonar-codes\KMeans.java	52	Field comments are required
164	C:\code-analysis\sonar-codes\KMeans.java	53	Field comments are required
165	C:\code-analysis\sonar-codes\KMeans.java	69	Parameter 'builder' is not assigned and could be declared final
166	C:\code-analysis\sonar-codes\KMeans.java	98	All classes and interfaces must belong to a named package
167	C:\code-analysis\sonar-codes\KMeans.java	100	Avoid variables with short names like k
168	C:\code-analysis\sonar-codes\KMeans.java	100	Field comments are required
169	C:\code-analysis\sonar-codes\KMeans.java	101	Field comments are required
170	C:\code-analysis\sonar-codes\KMeans.java	104	Field comments are required
171	C:\code-analysis\sonar-codes\KMeans.java	104	Field iterations has the same name as a method
172	C:\code-analysis\sonar-codes\KMeans.java	105	Avoid variables with short names like pp

173	C:\code-analysis\sonar-codes\KMeans.java	105	Field comments are required
174	C:\code-analysis\sonar-codes\KMeans.java	105	Field pp has the same name as a method
175	C:\code-analysis\sonar-codes\KMeans.java	106	Field comments are required
176	C:\code-analysis\sonar-codes\KMeans.java	106	Field epsilon has the same name as a method
177	C:\code-analysis\sonar-codes\KMeans.java	107	Field comments are required
178	C:\code-analysis\sonar-codes\KMeans.java	107	Field useEpsilon has the same name as a method
179	C:\code-analysis\sonar-codes\KMeans.java	108	Field comments are required
180	C:\code-analysis\sonar-codes\KMeans.java	108	Variables should start with a lowercase character, 'L1norm' starts with uppercase character.
181	C:\code-analysis\sonar-codes\KMeans.java	114	Avoid variables with short names like k
182	C:\code-analysis\sonar-codes\KMeans.java	114	Consider using varargs for methods or constructors which take an array the last parameter.
183	C:\code-analysis\sonar-codes\KMeans.java	114	Parameter 'k' is not assigned and could be declared final

184	C:\code-analysis\sonar-codes\KMeans.java	114	Parameter 'points' is not assigned and could be declared final
185	C:\code-analysis\sonar-codes\KMeans.java	114	The user-supplied array 'points' is stored directly.
186	C:\code-analysis\sonar-codes\KMeans.java	120	Local variable 'hashSet' could be declared final
187	C:\code-analysis\sonar-codes\KMeans.java	143	Parameter 'iterations' is not assigned and could be declared final
188	C:\code-analysis\sonar-codes\KMeans.java	144	Avoid using Literals in Conditional Statements
189	C:\code-analysis\sonar-codes\KMeans.java	153	Avoid using short method names
190	C:\code-analysis\sonar-codes\KMeans.java	153	Avoid variables with short names like pp
191	C:\code-analysis\sonar-codes\KMeans.java	153	Parameter 'pp' is not assigned and could be declared final
192	C:\code-analysis\sonar-codes\KMeans.java	161	Parameter 'epsilon' is not assigned and could be declared final
193	C:\code-analysis\sonar-codes\KMeans.java	162	Avoid using Literals in Conditional Statements
194	C:\code-analysis\sonar-codes\KMeans.java	172	Parameter 'useEpsilon' is not assigned and could be declared final

195	C:\code-analysis\sonar-codes\KMeans.java	180	Parameter 'L1norm' is not assigned and could be declared final
196	C:\code-analysis\sonar-codes\KMeans.java	180	Variables should start with a lowercase character, 'L1norm' starts with uppercase character.
197	C:\code-analysis\sonar-codes\KMeans.java	204	Found 'DD'-anomaly for variable 'bestCentroids' (lines '204'-'214').
198	C:\code-analysis\sonar-codes\KMeans.java	205	Found 'DD'-anomaly for variable 'bestAssignment' (lines '205'-'215').
199	C:\code-analysis\sonar-codes\KMeans.java	214	Found 'DD'-anomaly for variable 'bestCentroids' (lines '214'-'214').
200	C:\code-analysis\sonar-codes\KMeans.java	215	Found 'DD'-anomaly for variable 'bestAssignment' (lines '215'-'215').
201	C:\code-analysis\sonar-codes\KMeans.java	256	Found 'DD'-anomaly for variable 'minLocation' (lines '256'-'262').
202	C:\code-analysis\sonar-codes\KMeans.java	257	Found 'DD'-anomaly for variable 'minValue' (lines '257'-'257').
203	C:\code-analysis\sonar-codes\KMeans.java	257	Found 'DU'-anomaly for variable 'minValue' (lines '257'-'269').
204	C:\code-analysis\sonar-codes\KMeans.java	259	Potential violation of Law of Demeter (method chain calls)
205	C:\code-analysis\sonar-codes\KMeans.java	259	Potential violation of Law of Demeter (method chain calls)

206	C:\code-analysis\sonar-codes\KMeans.java	261	Found 'DD'-anomaly for variable 'minValue' (lines '261'-'257').
207	C:\code-analysis\sonar-codes\KMeans.java	261	Found 'DU'-anomaly for variable 'minValue' (lines '261'-'269').
208	C:\code-analysis\sonar-codes\KMeans.java	262	Found 'DD'-anomaly for variable 'minLocation' (lines '262'-'262').
209	C:\code-analysis\sonar-codes\KMeans.java	275	The method 'updateStep' has a Modified Cyclomatic Complexity of 14.
210	C:\code-analysis\sonar-codes\KMeans.java	275	The method 'updateStep' has a Standard Cyclomatic Complexity of 14.
211	C:\code-analysis\sonar-codes\KMeans.java	275	The method 'updateStep()' has a cyclomatic complexity of 14.
212	C:\code-analysis\sonar-codes\KMeans.java	275	The method 'updateStep()' has an NPath complexity of 468
213	C:\code-analysis\sonar-codes\KMeans.java	281	Found 'DD'-anomaly for variable 'clustSize' (lines '281'-'285').
214	C:\code-analysis\sonar-codes\KMeans.java	285	Found 'DD'-anomaly for variable 'clustSize' (lines '285'-'285').
215	C:\code-analysis\sonar-codes\KMeans.java	285	Found 'DU'-anomaly for variable 'clustSize' (lines '285'-'323').
216	C:\code-analysis\sonar-codes\KMeans.java	291	Local variable 'emptyCentroids' could be declared final

217	C:\code-analysis\sonar-codes\KMeans.java	303	Comment is too large: Line too long
218	C:\code-analysis\sonar-codes\KMeans.java	304	Substitute calls to size() == 0 (or size() != 0, size() > 0, size() < 1) with calls to isEmpty()
219	C:\code-analysis\sonar-codes\KMeans.java	305	Found 'DU'-anomaly for variable 'nonemptyCentroids' (lines '305'-'323').
220	C:\code-analysis\sonar-codes\KMeans.java	305	Local variable 'nonemptyCentroids' could be declared final
221	C:\code-analysis\sonar-codes\KMeans.java	308	Potential violation of Law of Demeter (method chain calls)
222	C:\code-analysis\sonar-codes\KMeans.java	310	Avoid variables with short names like r
223	C:\code-analysis\sonar-codes\KMeans.java	310	Found 'DU'-anomaly for variable 'r' (lines '310'-'323').
224	C:\code-analysis\sonar-codes\KMeans.java	310	Local variable 'r' could be declared final
225	C:\code-analysis\sonar-codes\KMeans.java	311	Local variable 'i' could be declared final
226	C:\code-analysis\sonar-codes\KMeans.java	313	Local variable 'rand' could be declared final
227	C:\code-analysis\sonar-codes\KMeans.java	314	Potential violation of Law of Demeter (method chain calls)

228	C:\code-analysis\sonar-codes\KMeans.java	315	Potential violation of Law of Demeter (method chain calls)
229	C:\code-analysis\sonar-codes\KMeans.java	330	Comment is too large: Line too long
230	C:\code-analysis\sonar-codes\KMeans.java	344	Found 'DU'-anomaly for variable 'copy' (lines '344'-'356').
231	C:\code-analysis\sonar-codes\KMeans.java	346	Found 'DU'-anomaly for variable 'gen' (lines '346'-'356').
232	C:\code-analysis\sonar-codes\KMeans.java	346	Local variable 'gen' could be declared final
233	C:\code-analysis\sonar-codes\KMeans.java	350	Found 'DD'-anomaly for variable 'rand' (lines '350'-'350').
234	C:\code-analysis\sonar-codes\KMeans.java	350	Found 'DU'-anomaly for variable 'rand' (lines '350'-'356').
235	C:\code-analysis\sonar-codes\KMeans.java	359	Comment is too large: Line too long
236	C:\code-analysis\sonar-codes\KMeans.java	364	The method 'plusplus' has a Modified Cyclomatic Complexity of 10.
237	C:\code-analysis\sonar-codes\KMeans.java	364	The method 'plusplus' has a Standard Cyclomatic Complexity of 10.
238	C:\code-analysis\sonar-codes\KMeans.java	364	The method 'plusplus()' has a cyclomatic complexity of 10.

239	C:\code-analysis\sonar-codes\KMeans.java	366	Avoid excessively long variable names like distToClosestCentroid
240	C:\code-analysis\sonar-codes\KMeans.java	366	Found 'DD'-anomaly for variable 'distToClosestCentroid' (lines '366'-'388').
241	C:\code-analysis\sonar-codes\KMeans.java	366	Found 'DU'-anomaly for variable 'distToClosestCentroid' (lines '366'-'420').
242	C:\code-analysis\sonar-codes\KMeans.java	367	Avoid excessively long variable names like weightedDistribution
243	C:\code-analysis\sonar-codes\KMeans.java	367	Found 'DD'-anomaly for variable 'weightedDistribution' (lines '367'-'397').
244	C:\code-analysis\sonar-codes\KMeans.java	367	Found 'DD'-anomaly for variable 'weightedDistribution' (lines '367'-'398').
245	C:\code-analysis\sonar-codes\KMeans.java	367	Found 'DU'-anomaly for variable 'weightedDistribution' (lines '367'-'420').
246	C:\code-analysis\sonar-codes\KMeans.java	369	Local variable 'gen' could be declared final
247	C:\code-analysis\sonar-codes\KMeans.java	370	Found 'DD'-anomaly for variable 'choose' (lines '370'-'376').
248	C:\code-analysis\sonar-codes\KMeans.java	370	Found 'DD'-anomaly for variable 'choose' (lines '370'-'408').
249	C:\code-analysis\sonar-codes\KMeans.java	376	Found 'DD'-anomaly for variable 'choose' (lines '376'-'376').

250	C:\code-analysis\sonar-codes\KMeans.java	376	Found 'DD'-anomaly for variable 'choose' (lines '376'-'408').
251	C:\code-analysis\sonar-codes\KMeans.java	376	Found 'DD'-anomaly for variable 'choose' (lines '376'-'412').
252	C:\code-analysis\sonar-codes\KMeans.java	376	Found 'DU'-anomaly for variable 'choose' (lines '376'-'420').
253	C:\code-analysis\sonar-codes\KMeans.java	381	Comment is too large: Line too long
254	C:\code-analysis\sonar-codes\KMeans.java	383	Comment is too large: Line too long
255	C:\code-analysis\sonar-codes\KMeans.java	384	Local variable 'tempDistance' could be declared final
256	C:\code-analysis\sonar-codes\KMeans.java	384	Potential violation of Law of Demeter (method chain calls)
257	C:\code-analysis\sonar-codes\KMeans.java	384	Potential violation of Law of Demeter (method chain calls)
258	C:\code-analysis\sonar-codes\KMeans.java	387	Avoid using Literals in Conditional Statements
259	C:\code-analysis\sonar-codes\KMeans.java	397	Found 'DD'-anomaly for variable 'weightedDistribution' (lines '397'-'397').
260	C:\code-analysis\sonar-codes\KMeans.java	397	Found 'DD'-anomaly for variable 'weightedDistribution' (lines '397'-'398').

261	C:\code-analysis\sonar-codes\KMeans.java	397	Found 'DU'-anomaly for variable 'weightedDistribution' (lines '397'-'420').
262	C:\code-analysis\sonar-codes\KMeans.java	403	Found 'DU'-anomaly for variable 'rand' (lines '403'-'420').
263	C:\code-analysis\sonar-codes\KMeans.java	403	Local variable 'rand' could be declared final
264	C:\code-analysis\sonar-codes\KMeans.java	408	Found 'DD'-anomaly for variable 'choose' (lines '408'-'376').
265	C:\code-analysis\sonar-codes\KMeans.java	408	Found 'DD'-anomaly for variable 'choose' (lines '408'-'408').
266	C:\code-analysis\sonar-codes\KMeans.java	408	Found 'DD'-anomaly for variable 'choose' (lines '408'-'412').
267	C:\code-analysis\sonar-codes\KMeans.java	408	Found 'DU'-anomaly for variable 'choose' (lines '408'-'420').
268	C:\code-analysis\sonar-codes\KMeans.java	411	Comment is too large: Line too long
269	C:\code-analysis\sonar-codes\KMeans.java	412	Found 'DD'-anomaly for variable 'choose' (lines '412'-'408').
270	C:\code-analysis\sonar-codes\KMeans.java	412	Found 'DD'-anomaly for variable 'choose' (lines '412'-'412').
271	C:\code-analysis\sonar-codes\KMeans.java	412	Found 'DU'-anomaly for variable 'choose' (lines '412'-'420').

272	C:\code-analysis\sonar-codes\KMeans.java	432	Parameter 'prevWCSS' is not assigned and could be declared final
273	C:\code-analysis\sonar-codes\KMeans.java	434	A method should have only one exit point, and that should be the last statement in the method
274	C:\code-analysis\sonar-codes\KMeans.java	438	Comment is too large: Line too long
275	C:\code-analysis\sonar-codes\KMeans.java	447	Parameter 'prevWCSS' is not assigned and could be declared final
276	C:\code-analysis\sonar-codes\KMeans.java	460	Avoid variables with short names like x
277	C:\code-analysis\sonar-codes\KMeans.java	460	Avoid variables with short names like y
278	C:\code-analysis\sonar-codes\KMeans.java	460	Consider using varargs for methods or constructors which take an array the last parameter.
279	C:\code-analysis\sonar-codes\KMeans.java	460	Parameter 'x' is not assigned and could be declared final
280	C:\code-analysis\sonar-codes\KMeans.java	460	Parameter 'y' is not assigned and could be declared final
281	C:\code-analysis\sonar-codes\KMeans.java	464	All classes and interfaces must belong to a named package
282	C:\code-analysis\sonar-codes\KMeans.java	464	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or

			make the class abstract to silence this warning.
283	C:\code-analysis\sonar-codes\KMeans.java	464	Header comments are required
284	C:\code-analysis\sonar-codes\KMeans.java	464	The utility class name 'Distance' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'
285	C:\code-analysis\sonar-codes\KMeans.java	473	Avoid using short method names
286	C:\code-analysis\sonar-codes\KMeans.java	473	Avoid variables with short names like x
287	C:\code-analysis\sonar-codes\KMeans.java	473	Avoid variables with short names like y
288	C:\code-analysis\sonar-codes\KMeans.java	473	Consider using varargs for methods or constructors which take an array the last parameter.
289	C:\code-analysis\sonar-codes\KMeans.java	473	Parameter 'x' is not assigned and could be declared final
290	C:\code-analysis\sonar-codes\KMeans.java	473	Parameter 'y' is not assigned and could be declared final
291	C:\code-analysis\sonar-codes\KMeans.java	473	The static method name 'L1' doesn't match '[a-z][a-zA-Z0-9]*'
292	C:\code-analysis\sonar-codes\KMeans.java	477	Potential violation of Law of Demeter (method chain calls)

293	C:\code-analysis\sonar-codes\KMeans.java	477	Potential violation of Law of Demeter (method chain calls)
294	C:\code-analysis\sonar-codes\KMeans.java	488	Avoid using short method names
295	C:\code-analysis\sonar-codes\KMeans.java	488	Avoid variables with short names like x
296	C:\code-analysis\sonar-codes\KMeans.java	488	Avoid variables with short names like y
297	C:\code-analysis\sonar-codes\KMeans.java	488	Consider using varargs for methods or constructors which take an array the last parameter.
298	C:\code-analysis\sonar-codes\KMeans.java	488	Parameter 'x' is not assigned and could be declared final
299	C:\code-analysis\sonar-codes\KMeans.java	488	Parameter 'y' is not assigned and could be declared final
300	C:\code-analysis\sonar-codes\KMeans.java	488	The static method name 'L2' doesn't match '[a-z][a-zA-Z0-9]*'
301	C:\code-analysis\sonar-codes\KMeans.java	492	Potential violation of Law of Demeter (method chain calls)
302	C:\code-analysis\sonar-codes\KMeans.java	492	Potential violation of Law of Demeter (method chain calls)
303	C:\code-analysis\sonar-codes\KMeans.java	492	Potential violation of Law of Demeter (method chain calls)

304	C:\code-analysis\sonar-codes\KMeans.java	492	Potential violation of Law of Demeter (method chain calls)
305	C:\code-analysis\sonar-codes\KMeans.java	498	Comment is too large: Line too long
306	C:\code-analysis\sonar-codes\KMeans.java	501	Variables should start with a lowercase character, 'WCSS' starts with uppercase character.
307	C:\code-analysis\sonar-codes\KMeans.java	506	Potential violation of Law of Demeter (method chain calls)
308	C:\code-analysis\sonar-codes\KMeans.java	506	Potential violation of Law of Demeter (method chain calls)
309	C:\code-analysis\sonar-codes\KMeans.java	516	Returning 'assignment' may expose an internal array.
310	C:\code-analysis\sonar-codes\KMeans.java	520	Returning 'centroids' may expose an internal array.
311	C:\code-analysis\sonar-codes\KMeans.java	527	Public method and constructor comments are required
312	C:\code-analysis\sonar-codes\KMeans.java	537	Parameter 'args' is not assigned and could be declared final
313	C:\code-analysis\sonar-codes\KMeans.java	540	Local variable 'data' could be declared final
314	C:\code-analysis\sonar-codes\KMeans.java	541	Local variable 'numPoints' could be declared final

315	C:\code-analysis\sonar-codes\KMeans.java	542	Local variable 'dimensions' could be declared final
316	C:\code-analysis\sonar-codes\KMeans.java	543	Avoid variables with short names like k
317	C:\code-analysis\sonar-codes\KMeans.java	543	Local variable 'k' could be declared final
318	C:\code-analysis\sonar-codes\KMeans.java	544	Local variable 'points' could be declared final
319	C:\code-analysis\sonar-codes\KMeans.java	548	Local variable 'clustering' could be declared final
320	C:\code-analysis\sonar-codes\KMeans.java	562	Found 'DU'-anomaly for variable 'centroids' (lines '562'-'576').
321	C:\code-analysis\sonar-codes\KMeans.java	562	Local variable 'centroids' could be declared final
322	C:\code-analysis\sonar-codes\KMeans.java	563	Local variable 'WCSS' could be declared final
323	C:\code-analysis\sonar-codes\KMeans.java	563	Variables should start with a lowercase character, 'WCSS' starts with uppercase character.
324	C:\code-analysis\sonar-codes\KMeans.java	568	Potential violation of Law of Demeter (method chain calls)
325	C:\code-analysis\sonar-codes\KMeans.java	568	Potential violation of Law of Demeter (method chain calls)

326	C:\code-analysis\sonar-codes\KMeans.java	568	Potential violation of Law of Demeter (method chain calls)
327	C:\code-analysis\sonar-codes\KMeans.java	568	Potential violation of Law of Demeter (method chain calls)
328	C:\code-analysis\sonar-codes\Kruskal.java	4	All classes and interfaces must belong to a named package
329	C:\code-analysis\sonar-codes\Kruskal.java	4	Each class should declare at least one constructor
330	C:\code-analysis\sonar-codes\Kruskal.java	4	Header comments are required
331	C:\code-analysis\sonar-codes\Kruskal.java	5	Parameter 'args' is not assigned and could be declared final
332	C:\code-analysis\sonar-codes\Kruskal.java	5	Public method and constructor comments are required
333	C:\code-analysis\sonar-codes\Kruskal.java	6	Local variable 'graphEdges' could be declared final
334	C:\code-analysis\sonar-codes\Kruskal.java	20	Comment is too large: Line too long
335	C:\code-analysis\sonar-codes\Kruskal.java	23	Comment is too large: Line too long
336	C:\code-analysis\sonar-codes\Kruskal.java	23	Local variable 'nodeCount' could be declared final

337	C:\code-analysis\sonar-codes\Kruskal.java	25	Comment is too large: Line too long
338	C:\code-analysis\sonar-codes\Kruskal.java	25	Local variable 'graph' could be declared final
339	C:\code-analysis\sonar-codes\Kruskal.java	29	Avoid using implementation types like 'ArrayList'; use the interface instead
340	C:\code-analysis\sonar-codes\Kruskal.java	29	Avoid using implementation types like 'ArrayList'; use the interface instead
341	C:\code-analysis\sonar-codes\Kruskal.java	29	Found 'UR'-anomaly for variable 'edge' (lines '29'-'54').
342	C:\code-analysis\sonar-codes\Kruskal.java	29	Parameter 'graphEdges' is not assigned and could be declared final
343	C:\code-analysis\sonar-codes\Kruskal.java	29	Parameter 'nodeCount' is not assigned and could be declared final
344	C:\code-analysis\sonar-codes\Kruskal.java	29	Public method and constructor comments are required
345	C:\code-analysis\sonar-codes\Kruskal.java	33	Local variable 'mstEdges' could be declared final
346	C:\code-analysis\sonar-codes\Kruskal.java	35	Comment is too large: Line too long
347	C:\code-analysis\sonar-codes\Kruskal.java	35	Found 'DU'-anomaly for variable 'nodeSet' (lines '35'-'67').

348	C:\code-analysis\sonar-codes\Kruskal.java	35	Local variable 'nodeSet' could be declared final
349	C:\code-analysis\sonar-codes\Kruskal.java	37	Comment is too large: Line too long
350	C:\code-analysis\sonar-codes\Kruskal.java	38	Local variable 'currentEdge' could be declared final
351	C:\code-analysis\sonar-codes\Kruskal.java	39	Local variable 'root1' could be declared final
352	C:\code-analysis\sonar-codes\Kruskal.java	39	Potential violation of Law of Demeter (object not created locally)
353	C:\code-analysis\sonar-codes\Kruskal.java	40	Local variable 'root2' could be declared final
354	C:\code-analysis\sonar-codes\Kruskal.java	40	Potential violation of Law of Demeter (object not created locally)
355	C:\code-analysis\sonar-codes\Kruskal.java	41	Potential violation of Law of Demeter (object not created locally)
356	C:\code-analysis\sonar-codes\Kruskal.java	41	Potential violation of Law of Demeter (object not created locally)
357	C:\code-analysis\sonar-codes\Kruskal.java	41	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
358	C:\code-analysis\sonar-codes\Kruskal.java	42	Found 'DD'-anomaly for variable 'unionMessage' (lines '42'-'46').

359	C:\code-analysis\sonar-codes\Kruskal.java	48	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
360	C:\code-analysis\sonar-codes\Kruskal.java	51	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
361	C:\code-analysis\sonar-codes\Kruskal.java	52	Avoid excessively long variable names like mstTotalEdgeWeight
362	C:\code-analysis\sonar-codes\Kruskal.java	53	Local variable 'edge' could be declared final
363	C:\code-analysis\sonar-codes\Kruskal.java	54	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
364	C:\code-analysis\sonar-codes\Kruskal.java	57	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
365	C:\code-analysis\sonar-codes\Kruskal.java	71	All classes and interfaces must belong to a named package
366	C:\code-analysis\sonar-codes\Kruskal.java	71	Avoid short class names like Edge
367	C:\code-analysis\sonar-codes\Kruskal.java	71	Header comments are required
368	C:\code-analysis\sonar-codes\Kruskal.java	72	Field comments are required
369	C:\code-analysis\sonar-codes\Kruskal.java	72	Private field 'vertex1' could be made final; it is only initialized in the declaration or constructor.

370	C:\code-analysis\sonar-codes\Kruskal.java	73	Field comments are required
371	C:\code-analysis\sonar-codes\Kruskal.java	73	Private field 'vertex2' could be made final; it is only initialized in the declaration or constructor.
372	C:\code-analysis\sonar-codes\Kruskal.java	74	Field comments are required
373	C:\code-analysis\sonar-codes\Kruskal.java	74	Private field 'weight' could be made final; it is only initialized in the declaration or constructor.
374	C:\code-analysis\sonar-codes\Kruskal.java	76	Parameter 'vertex1' is not assigned and could be declared final
375	C:\code-analysis\sonar-codes\Kruskal.java	76	Parameter 'vertex2' is not assigned and could be declared final
376	C:\code-analysis\sonar-codes\Kruskal.java	76	Parameter 'weight' is not assigned and could be declared final
377	C:\code-analysis\sonar-codes\Kruskal.java	76	Public method and constructor comments are required
378	C:\code-analysis\sonar-codes\Kruskal.java	95	Parameter 'otherEdge' is not assigned and could be declared final
379	C:\code-analysis\sonar-codes\Kruskal.java	122	All classes and interfaces must belong to a named package
380	C:\code-analysis\sonar-codes\Kruskal.java	123	Field comments are required

381	C:\code-analysis\sonar-codes\Kruskal.java	126	Returning 'set' may expose an internal array.
382	C:\code-analysis\sonar-codes\Kruskal.java	133	Parameter 'numElements' is not assigned and could be declared final
383	C:\code-analysis\sonar-codes\Kruskal.java	147	Parameter 'root1' is not assigned and could be declared final
384	C:\code-analysis\sonar-codes\Kruskal.java	147	Parameter 'root2' is not assigned and could be declared final
385	C:\code-analysis\sonar-codes\Kruskal.java	165	Avoid variables with short names like x
386	C:\code-analysis\sonar-codes\Kruskal.java	165	Parameter 'x' is not assigned and could be declared final
387	C:\code-analysis\sonar-codes\Kruskal.java	167	A method should have only one exit point, and that should be the last statement in the method
388	C:\code-analysis\sonar-codes\Layer.java	4	All classes and interfaces must belong to a named package
389	C:\code-analysis\sonar-codes\Layer.java	4	Header comments are required
390	C:\code-analysis\sonar-codes\Layer.java	7	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'n_neurons' is not final.
391	C:\code-analysis\sonar-codes\Layer.java	7	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'prev_n_neurons' is not final.

392	C:\code-analysis\sonar-codes\Layer.java	7	Parameter 'n_neurons' is not assigned and could be declared final
393	C:\code-analysis\sonar-codes\Layer.java	7	Parameter 'prev_n_neurons' is not assigned and could be declared final
394	C:\code-analysis\sonar-codes\Layer.java	7	Parameter 'rand' is not assigned and could be declared final
395	C:\code-analysis\sonar-codes\Layer.java	7	Public method and constructor comments are required
396	C:\code-analysis\sonar-codes\Layer.java	18	Avoid instantiating new objects inside loops
397	C:\code-analysis\sonar-codes\Layer.java	22	Avoid prefixing parameters by in, out or inOut. Uses Javadoc to document this behavior.
398	C:\code-analysis\sonar-codes\Layer.java	22	Avoid variables with short names like in
399	C:\code-analysis\sonar-codes\Layer.java	22	Consider using varargs for methods or constructors which take an array the last parameter.
400	C:\code-analysis\sonar-codes\Layer.java	22	Parameter 'in' is not assigned and could be declared final
401	C:\code-analysis\sonar-codes\Layer.java	22	Public method and constructor comments are required
402	C:\code-analysis\sonar-codes\Layer.java	22	The static method name 'add_bias' doesn't match '[a-z][a-zA-Z0-9]*'

403	C:\code-analysis\sonar-codes\Layer.java	24	Found 'DD'-anomaly for variable 'out' (lines '24'-'26').
404	C:\code-analysis\sonar-codes\Layer.java	24	Found 'DD'-anomaly for variable 'out' (lines '24'-'27').
405	C:\code-analysis\sonar-codes\Layer.java	26	Found 'DD'-anomaly for variable 'out' (lines '26'-'26').
406	C:\code-analysis\sonar-codes\Layer.java	26	Found 'DD'-anomaly for variable 'out' (lines '26'-'27').
407	C:\code-analysis\sonar-codes\Layer.java	32	Avoid prefixing parameters by in, out or inOut. Uses Javadoc to document this behavior.
408	C:\code-analysis\sonar-codes\Layer.java	32	Avoid variables with short names like in
409	C:\code-analysis\sonar-codes\Layer.java	32	Consider using varargs for methods or constructors which take an array the last parameter.
410	C:\code-analysis\sonar-codes\Layer.java	32	Parameter 'in' is not assigned and could be declared final
411	C:\code-analysis\sonar-codes\Layer.java	32	Public method and constructor comments are required
412	C:\code-analysis\sonar-codes\Layer.java	37	Avoid if (x != y) ..; else ..;
413	C:\code-analysis\sonar-codes\Layer.java	37	Potential violation of Law of Demeter (method chain calls)

414	C:\code-analysis\sonar-codes\Layer.java	42	Potential violation of Law of Demeter (method chain calls)
415	C:\code-analysis\sonar-codes\Layer.java	42	Useless parentheses.
416	C:\code-analysis\sonar-codes\Layer.java	42	Useless parentheses.
417	C:\code-analysis\sonar-codes\Layer.java	46	Potential violation of Law of Demeter (method chain calls)
418	C:\code-analysis\sonar-codes\Layer.java	51	Returning '_outputs' may expose an internal array.
419	C:\code-analysis\sonar-codes\Layer.java	54	Public method and constructor comments are required
420	C:\code-analysis\sonar-codes\Layer.java	55	Avoid variables with short names like i
421	C:\code-analysis\sonar-codes\Layer.java	55	Parameter 'i' is not assigned and could be declared final
422	C:\code-analysis\sonar-codes\Layer.java	55	Public method and constructor comments are required
423	C:\code-analysis\sonar-codes\Layer.java	56	Avoid variables with short names like i
424	C:\code-analysis\sonar-codes\Layer.java	56	Parameter 'i' is not assigned and could be declared final

425	C:\code-analysis\sonar-codes\Layer.java	56	Potential violation of Law of Demeter (method chain calls)
426	C:\code-analysis\sonar-codes\Layer.java	56	Public method and constructor comments are required
427	C:\code-analysis\sonar-codes\Layer.java	57	Avoid variables with short names like i
428	C:\code-analysis\sonar-codes\Layer.java	57	Parameter 'i' is not assigned and could be declared final
429	C:\code-analysis\sonar-codes\Layer.java	57	Potential violation of Law of Demeter (method chain calls)
430	C:\code-analysis\sonar-codes\Layer.java	57	Public method and constructor comments are required
431	C:\code-analysis\sonar-codes\Layer.java	58	Avoid variables with short names like i
432	C:\code-analysis\sonar-codes\Layer.java	58	Avoid variables with short names like j
433	C:\code-analysis\sonar-codes\Layer.java	58	Parameter 'i' is not assigned and could be declared final
434	C:\code-analysis\sonar-codes\Layer.java	58	Parameter 'j' is not assigned and could be declared final
435	C:\code-analysis\sonar-codes\Layer.java	58	Potential violation of Law of Demeter (method chain calls)

436	C:\code-analysis\sonar-codes\Layer.java	58	Public method and constructor comments are required
437	C:\code-analysis\sonar-codes\Layer.java	59	Avoid variables with short names like i
438	C:\code-analysis\sonar-codes\Layer.java	59	Avoid variables with short names like j
439	C:\code-analysis\sonar-codes\Layer.java	59	Avoid variables with short names like v
440	C:\code-analysis\sonar-codes\Layer.java	59	Parameter 'i' is not assigned and could be declared final
441	C:\code-analysis\sonar-codes\Layer.java	59	Parameter 'j' is not assigned and could be declared final
442	C:\code-analysis\sonar-codes\Layer.java	59	Parameter 'v' is not assigned and could be declared final
443	C:\code-analysis\sonar-codes\Layer.java	59	Potential violation of Law of Demeter (method chain calls)
444	C:\code-analysis\sonar-codes\Layer.java	59	Public method and constructor comments are required
445	C:\code-analysis\sonar-codes\Layer.java	62	Field comments are required
446	C:\code-analysis\sonar-codes\Layer.java	62	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.

447	C:\code-analysis\sonar-codes\Layer.java	62	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_n_neurons' is not final.
448	C:\code-analysis\sonar-codes\Layer.java	62	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_prev_n_neurons' is not final.
449	C:\code-analysis\sonar-codes\Layer.java	62	Perhaps '_prev_n_neurons' could be replaced by a local variable.
450	C:\code-analysis\sonar-codes\Layer.java	62	Private field '_n_neurons' could be made final; it is only initialized in the declaration or constructor.
451	C:\code-analysis\sonar-codes\Layer.java	62	Private field '_prev_n_neurons' could be made final; it is only initialized in the declaration or constructor.
452	C:\code-analysis\sonar-codes\Layer.java	63	Avoid using implementation types like 'ArrayList'; use the interface instead
453	C:\code-analysis\sonar-codes\Layer.java	63	Avoid using implementation types like 'ArrayList'; use the interface instead
454	C:\code-analysis\sonar-codes\Layer.java	63	Field comments are required
455	C:\code-analysis\sonar-codes\Layer.java	63	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
456	C:\code-analysis\sonar-codes\Layer.java	63	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_neurons' is not final.
457	C:\code-analysis\sonar-codes\Layer.java	63	Private field '_neurons' could be made final; it is only initialized in the declaration or constructor.

458	C:\code-analysis\sonar-codes\Layer.java	64	Field comments are required
459	C:\code-analysis\sonar-codes\Layer.java	64	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
460	C:\code-analysis\sonar-codes\Layer.java	64	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_outputs' is not final.
461	C:\code-analysis\sonar-codes\Metric.java	1	Comment is too large: Too many lines
462	C:\code-analysis\sonar-codes\Metric.java	19	Comment is too large: Too many lines
463	C:\code-analysis\sonar-codes\Metric.java	32	All classes and interfaces must belong to a named package
464	C:\code-analysis\sonar-codes\Mlp.java	6	All classes and interfaces must belong to a named package
465	C:\code-analysis\sonar-codes\Mlp.java	6	Avoid short class names like Mlp
466	C:\code-analysis\sonar-codes\Mlp.java	6	Header comments are required
467	C:\code-analysis\sonar-codes\Mlp.java	9	Consider using varargs for methods or constructors which take an array the last parameter.
468	C:\code-analysis\sonar-codes\Mlp.java	9	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'nn_neurons' is not final.

469	C:\code-analysis\sonar-codes\Mlp.java	9	Parameter 'nn_neurons' is not assigned and could be declared final
470	C:\code-analysis\sonar-codes\Mlp.java	9	Public method and constructor comments are required
471	C:\code-analysis\sonar-codes\Mlp.java	11	Local variable 'rand' could be declared final
472	C:\code-analysis\sonar-codes\Mlp.java	17	Avoid instantiating new objects inside loops
473	C:\code-analysis\sonar-codes\Mlp.java	25	Avoid instantiating new objects inside loops
474	C:\code-analysis\sonar-codes\Mlp.java	32	Avoid instantiating new objects inside loops
475	C:\code-analysis\sonar-codes\Mlp.java	35	Avoid reassigning parameters such as 'inputs'
476	C:\code-analysis\sonar-codes\Mlp.java	35	Consider using varargs for methods or constructors which take an array the last parameter.
477	C:\code-analysis\sonar-codes\Mlp.java	35	Public method and constructor comments are required
478	C:\code-analysis\sonar-codes\Mlp.java	39	Useless parentheses.
479	C:\code-analysis\sonar-codes\Mlp.java	39	Useless parentheses.

480	C:\code-analysis\sonar-codes\Mlp.java	41	Found 'DD'-anomaly for variable 'outputs' (lines '41'-'44').
481	C:\code-analysis\sonar-codes\Mlp.java	44	Potential violation of Law of Demeter (method chain calls)
482	C:\code-analysis\sonar-codes\Mlp.java	51	Consider using varargs for methods or constructors which take an array the last parameter.
483	C:\code-analysis\sonar-codes\Mlp.java	51	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'desired_output' is not final.
484	C:\code-analysis\sonar-codes\Mlp.java	51	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'nn_output' is not final.
485	C:\code-analysis\sonar-codes\Mlp.java	51	Parameter 'desired_output' is not assigned and could be declared final
486	C:\code-analysis\sonar-codes\Mlp.java	51	Parameter 'nn_output' is not assigned and could be declared final
487	C:\code-analysis\sonar-codes\Mlp.java	53	Avoid variables with short names like d
488	C:\code-analysis\sonar-codes\Mlp.java	56	Avoid if (x != y) ..; else ..;
489	C:\code-analysis\sonar-codes\Mlp.java	61	Useless parentheses.
490	C:\code-analysis\sonar-codes\Mlp.java	61	Useless parentheses.

491	C:\code-analysis\sonar-codes\Mlp.java	63	Avoid variables with short names like e
492	C:\code-analysis\sonar-codes\Mlp.java	70	Avoid using implementation types like 'ArrayList'; use the interface instead
493	C:\code-analysis\sonar-codes\Mlp.java	70	Avoid using implementation types like 'ArrayList'; use the interface instead
494	C:\code-analysis\sonar-codes\Mlp.java	70	Parameter 'examples' is not assigned and could be declared final
495	C:\code-analysis\sonar-codes\Mlp.java	70	Public method and constructor comments are required
496	C:\code-analysis\sonar-codes\Mlp.java	71	Avoid using implementation types like 'ArrayList'; use the interface instead
497	C:\code-analysis\sonar-codes\Mlp.java	71	Avoid using implementation types like 'ArrayList'; use the interface instead
498	C:\code-analysis\sonar-codes\Mlp.java	71	Parameter 'results' is not assigned and could be declared final
499	C:\code-analysis\sonar-codes\Mlp.java	75	Useless parentheses.
500	C:\code-analysis\sonar-codes\Mlp.java	75	Useless parentheses.
501	C:\code-analysis\sonar-codes\Mlp.java	77	Avoid variables with short names like e

502	C:\code-analysis\sonar-codes\Mlp.java	86	Consider using varargs for methods or constructors which take an array the last parameter.
503	C:\code-analysis\sonar-codes\Mlp.java	86	Parameter 'results' is not assigned and could be declared final
504	C:\code-analysis\sonar-codes\Mlp.java	90	Potential violation of Law of Demeter (method chain calls)
505	C:\code-analysis\sonar-codes\Mlp.java	93	Potential violation of Law of Demeter (method chain calls)
506	C:\code-analysis\sonar-codes\Mlp.java	94	Potential violation of Law of Demeter (method chain calls)
507	C:\code-analysis\sonar-codes\Mlp.java	95	Potential violation of Law of Demeter (method chain calls)
508	C:\code-analysis\sonar-codes\Mlp.java	99	Potential violation of Law of Demeter (method chain calls)
509	C:\code-analysis\sonar-codes\Mlp.java	100	Potential violation of Law of Demeter (method chain calls)
510	C:\code-analysis\sonar-codes\Mlp.java	100	Potential violation of Law of Demeter (method chain calls)
511	C:\code-analysis\sonar-codes\Mlp.java	101	Potential violation of Law of Demeter (method chain calls)
512	C:\code-analysis\sonar-codes\Mlp.java	101	Potential violation of Law of Demeter (method chain calls)

513	C:\code-analysis\sonar-codes\Mlp.java	111	Potential violation of Law of Demeter (method chain calls)
514	C:\code-analysis\sonar-codes\Mlp.java	112	Local variable 'weights' could be declared final
515	C:\code-analysis\sonar-codes\Mlp.java	112	Potential violation of Law of Demeter (method chain calls)
516	C:\code-analysis\sonar-codes\Mlp.java	114	Potential violation of Law of Demeter (method chain calls)
517	C:\code-analysis\sonar-codes\Mlp.java	114	Potential violation of Law of Demeter (method chain calls)
518	C:\code-analysis\sonar-codes\Mlp.java	123	Potential violation of Law of Demeter (method chain calls)
519	C:\code-analysis\sonar-codes\Mlp.java	124	Local variable 'weights' could be declared final
520	C:\code-analysis\sonar-codes\Mlp.java	124	Potential violation of Law of Demeter (method chain calls)
521	C:\code-analysis\sonar-codes\Mlp.java	126	Potential violation of Law of Demeter (method chain calls)
522	C:\code-analysis\sonar-codes\Mlp.java	126	Potential violation of Law of Demeter (method chain calls)
523	C:\code-analysis\sonar-codes\Mlp.java	126	Potential violation of Law of Demeter (method chain calls)

524	C:\code-analysis\sonar-codes\Mlp.java	127	Potential violation of Law of Demeter (method chain calls)
525	C:\code-analysis\sonar-codes\Mlp.java	132	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'learning_rate' is not final.
526	C:\code-analysis\sonar-codes\Mlp.java	132	Parameter 'learning_rate' is not assigned and could be declared final
527	C:\code-analysis\sonar-codes\Mlp.java	135	Potential violation of Law of Demeter (method chain calls)
528	C:\code-analysis\sonar-codes\Mlp.java	136	Local variable 'weights' could be declared final
529	C:\code-analysis\sonar-codes\Mlp.java	136	Potential violation of Law of Demeter (method chain calls)
530	C:\code-analysis\sonar-codes\Mlp.java	138	Potential violation of Law of Demeter (method chain calls)
531	C:\code-analysis\sonar-codes\Mlp.java	138	Potential violation of Law of Demeter (method chain calls)
532	C:\code-analysis\sonar-codes\Mlp.java	138	Potential violation of Law of Demeter (method chain calls)
533	C:\code-analysis\sonar-codes\Mlp.java	138	Potential violation of Law of Demeter (method chain calls)
534	C:\code-analysis\sonar-codes\Mlp.java	138	Potential violation of Law of Demeter (method chain calls)

535	C:\code-analysis\sonar-codes\Mlp.java	139	Potential violation of Law of Demeter (method chain calls)
536	C:\code-analysis\sonar-codes\Mlp.java	139	Potential violation of Law of Demeter (method chain calls)
537	C:\code-analysis\sonar-codes\Mlp.java	139	Potential violation of Law of Demeter (method chain calls)
538	C:\code-analysis\sonar-codes\Mlp.java	139	Potential violation of Law of Demeter (method chain calls)
539	C:\code-analysis\sonar-codes\Mlp.java	144	Avoid using implementation types like 'ArrayList'; use the interface instead
540	C:\code-analysis\sonar-codes\Mlp.java	144	Avoid using implementation types like 'ArrayList'; use the interface instead
541	C:\code-analysis\sonar-codes\Mlp.java	144	Parameter 'examples' is not assigned and could be declared final
542	C:\code-analysis\sonar-codes\Mlp.java	145	Avoid using implementation types like 'ArrayList'; use the interface instead
543	C:\code-analysis\sonar-codes\Mlp.java	145	Avoid using implementation types like 'ArrayList'; use the interface instead
544	C:\code-analysis\sonar-codes\Mlp.java	145	Parameter 'results' is not assigned and could be declared final
545	C:\code-analysis\sonar-codes\Mlp.java	146	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'learning_rate' is not final.

546	C:\code-analysis\sonar-codes\Mlp.java	146	Parameter 'learning_rate' is not assigned and could be declared final
547	C:\code-analysis\sonar-codes\Mlp.java	159	Avoid using implementation types like 'ArrayList'; use the interface instead
548	C:\code-analysis\sonar-codes\Mlp.java	159	Avoid using implementation types like 'ArrayList'; use the interface instead
549	C:\code-analysis\sonar-codes\Mlp.java	159	Parameter 'examples' is not assigned and could be declared final
550	C:\code-analysis\sonar-codes\Mlp.java	159	Public method and constructor comments are required
551	C:\code-analysis\sonar-codes\Mlp.java	160	Avoid using implementation types like 'ArrayList'; use the interface instead
552	C:\code-analysis\sonar-codes\Mlp.java	160	Avoid using implementation types like 'ArrayList'; use the interface instead
553	C:\code-analysis\sonar-codes\Mlp.java	160	Parameter 'results' is not assigned and could be declared final
554	C:\code-analysis\sonar-codes\Mlp.java	161	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'learning_rate' is not final.
555	C:\code-analysis\sonar-codes\Mlp.java	161	Parameter 'learning_rate' is not assigned and could be declared final
556	C:\code-analysis\sonar-codes\Mlp.java	164	Useless parentheses.

557	C:\code-analysis\sonar-codes\Mlp.java	164	Useless parentheses.
558	C:\code-analysis\sonar-codes\Mlp.java	166	Avoid variables with short names like e
559	C:\code-analysis\sonar-codes\Mlp.java	176	Avoid using implementation types like 'ArrayList'; use the interface instead
560	C:\code-analysis\sonar-codes\Mlp.java	176	Avoid using implementation types like 'ArrayList'; use the interface instead
561	C:\code-analysis\sonar-codes\Mlp.java	176	Field comments are required
562	C:\code-analysis\sonar-codes\Mlp.java	176	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
563	C:\code-analysis\sonar-codes\Mlp.java	176	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_layers' is not final.
564	C:\code-analysis\sonar-codes\Mlp.java	176	Private field '_layers' could be made final; it is only initialized in the declaration or constructor.
565	C:\code-analysis\sonar-codes\Mlp.java	177	Avoid using implementation types like 'ArrayList'; use the interface instead
566	C:\code-analysis\sonar-codes\Mlp.java	177	Avoid using implementation types like 'ArrayList'; use the interface instead
567	C:\code-analysis\sonar-codes\Mlp.java	177	Field comments are required

568	C:\code-analysis\sonar-codes\Mlp.java	177	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
569	C:\code-analysis\sonar-codes\Mlp.java	177	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_delta_w' is not final.
570	C:\code-analysis\sonar-codes\Mlp.java	178	Avoid using implementation types like 'ArrayList'; use the interface instead
571	C:\code-analysis\sonar-codes\Mlp.java	178	Avoid using implementation types like 'ArrayList'; use the interface instead
572	C:\code-analysis\sonar-codes\Mlp.java	178	Field comments are required
573	C:\code-analysis\sonar-codes\Mlp.java	178	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
574	C:\code-analysis\sonar-codes\Mlp.java	178	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_grad_ex' is not final.
575	C:\code-analysis\sonar-codes\Mlp.java	178	Private field '_grad_ex' could be made final; it is only initialized in the declaration or constructor.
576	C:\code-analysis\sonar-codes\Mlp.java	184	Parameter 'args' is not assigned and could be declared final
577	C:\code-analysis\sonar-codes\Mlp.java	187	Avoid variables with short names like ex
578	C:\code-analysis\sonar-codes\Mlp.java	187	Local variable 'ex' could be declared final

579	C:\code-analysis\sonar-codes\Mlp.java	188	Local variable 'out' could be declared final
580	C:\code-analysis\sonar-codes\Mlp.java	190	Avoid instantiating new objects inside loops
581	C:\code-analysis\sonar-codes\Mlp.java	191	Avoid instantiating new objects inside loops
582	C:\code-analysis\sonar-codes\Mlp.java	195	Potential violation of Law of Demeter (method chain calls)
583	C:\code-analysis\sonar-codes\Mlp.java	195	Potential violation of Law of Demeter (method chain calls)
584	C:\code-analysis\sonar-codes\Mlp.java	195	Potential violation of Law of Demeter (method chain calls)
585	C:\code-analysis\sonar-codes\Mlp.java	196	Potential violation of Law of Demeter (method chain calls)
586	C:\code-analysis\sonar-codes\Mlp.java	196	Potential violation of Law of Demeter (method chain calls)
587	C:\code-analysis\sonar-codes\Mlp.java	196	Potential violation of Law of Demeter (method chain calls)
588	C:\code-analysis\sonar-codes\Mlp.java	197	Potential violation of Law of Demeter (method chain calls)
589	C:\code-analysis\sonar-codes\Mlp.java	197	Potential violation of Law of Demeter (method chain calls)

590	C:\code-analysis\sonar-codes\Mlp.java	197	Potential violation of Law of Demeter (method chain calls)
591	C:\code-analysis\sonar-codes\Mlp.java	198	Potential violation of Law of Demeter (method chain calls)
592	C:\code-analysis\sonar-codes\Mlp.java	198	Potential violation of Law of Demeter (method chain calls)
593	C:\code-analysis\sonar-codes\Mlp.java	198	Potential violation of Law of Demeter (method chain calls)
594	C:\code-analysis\sonar-codes\Mlp.java	200	Local variable 'nn_neurons' could be declared final
595	C:\code-analysis\sonar-codes\Mlp.java	200	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'nn_neurons' is not final.
596	C:\code-analysis\sonar-codes\Mlp.java	201	Potential violation of Law of Demeter (method chain calls)
597	C:\code-analysis\sonar-codes\Mlp.java	202	Potential violation of Law of Demeter (method chain calls)
598	C:\code-analysis\sonar-codes\Mlp.java	206	Found 'DU'-anomaly for variable 'mlp' (lines '206'-'223').
599	C:\code-analysis\sonar-codes\Mlp.java	206	Local variable 'mlp' could be declared final
600	C:\code-analysis\sonar-codes\Mlp.java	209	Local variable 'fout' could be declared final

601	C:\code-analysis\sonar-codes\Mlp.java	214	Local variable 'error' could be declared final
602	C:\code-analysis\sonar-codes\MyFrame.java	5	All classes and interfaces must belong to a named package
603	C:\code-analysis\sonar-codes\MyFrame.java	5	Header comments are required
604	C:\code-analysis\sonar-codes\MyFrame.java	7	Field comments are required
605	C:\code-analysis\sonar-codes\MyFrame.java	7	Private field 'btnTutup' could be made final; it is only initialized in the declaration or constructor.
606	C:\code-analysis\sonar-codes\MyFrame.java	8	Field comments are required
607	C:\code-analysis\sonar-codes\MyFrame.java	8	Private field 'btnTambah' could be made final; it is only initialized in the declaration or constructor.
608	C:\code-analysis\sonar-codes\MyFrame.java	10	Field comments are required
609	C:\code-analysis\sonar-codes\MyFrame.java	10	Private field 'txtA' could be made final; it is only initialized in the declaration or constructor.
610	C:\code-analysis\sonar-codes\MyFrame.java	11	Field comments are required
611	C:\code-analysis\sonar-codes\MyFrame.java	11	Private field 'txtB' could be made final; it is only initialized in the declaration or constructor.

612	C:\code-analysis\sonar-codes\MyFrame.java	12	Field comments are required
613	C:\code-analysis\sonar-codes\MyFrame.java	12	Private field 'txtC' could be made final; it is only initialized in the declaration or constructor.
614	C:\code-analysis\sonar-codes\MyFrame.java	14	Field comments are required
615	C:\code-analysis\sonar-codes\MyFrame.java	14	Private field 'lblA' could be made final; it is only initialized in the declaration or constructor.
616	C:\code-analysis\sonar-codes\MyFrame.java	15	Field comments are required
617	C:\code-analysis\sonar-codes\MyFrame.java	15	Private field 'lblB' could be made final; it is only initialized in the declaration or constructor.
618	C:\code-analysis\sonar-codes\MyFrame.java	16	Field comments are required
619	C:\code-analysis\sonar-codes\MyFrame.java	16	Private field 'lblC' could be made final; it is only initialized in the declaration or constructor.
620	C:\code-analysis\sonar-codes\MyFrame.java	18	It is a good practice to call super() in a constructor
621	C:\code-analysis\sonar-codes\MyFrame.java	18	Public method and constructor comments are required
622	C:\code-analysis\sonar-codes\MyFrame.java	57	Avoid variables with short names like e

623	C:\code-analysis\sonar-codes\MyFrame.java	57	Parameter 'e' is not assigned and could be declared final
624	C:\code-analysis\sonar-codes\MyFrame.java	57	Public method and constructor comments are required
625	C:\code-analysis\sonar-codes\MyFrame.java	63	Avoid variables with short names like e
626	C:\code-analysis\sonar-codes\MyFrame.java	63	Parameter 'e' is not assigned and could be declared final
627	C:\code-analysis\sonar-codes\MyFrame.java	63	Public method and constructor comments are required
628	C:\code-analysis\sonar-codes\MyFrame.java	69	Avoid variables with short names like e
629	C:\code-analysis\sonar-codes\MyFrame.java	69	Parameter 'e' is not assigned and could be declared final
630	C:\code-analysis\sonar-codes\MyFrame.java	69	Public method and constructor comments are required
631	C:\code-analysis\sonar-codes\MyFrame.java	75	Avoid unused method parameters such as 'evt'.
632	C:\code-analysis\sonar-codes\MyFrame.java	75	Parameter 'evt' is not assigned and could be declared final
633	C:\code-analysis\sonar-codes\MyFrame.java	79	Avoid unused method parameters such as 'evt'.

634	C:\code-analysis\sonar-codes\MyFrame.java	79	Parameter 'evt' is not assigned and could be declared final
635	C:\code-analysis\sonar-codes\MyFrame.java	80	Avoid variables with short names like x
636	C:\code-analysis\sonar-codes\MyFrame.java	80	Avoid variables with short names like y
637	C:\code-analysis\sonar-codes\MyFrame.java	80	Avoid variables with short names like z
638	C:\code-analysis\sonar-codes\MyFrame.java	80	Use one line for each declaration, it enhances code readability.
639	C:\code-analysis\sonar-codes\Neuron.java	2	All classes and interfaces must belong to a named package
640	C:\code-analysis\sonar-codes\Neuron.java	2	Header comments are required
641	C:\code-analysis\sonar-codes\Neuron.java	5	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), 'prev_n_neurons' is not final.
642	C:\code-analysis\sonar-codes\Neuron.java	5	Parameter 'prev_n_neurons' is not assigned and could be declared final
643	C:\code-analysis\sonar-codes\Neuron.java	5	Parameter 'rand' is not assigned and could be declared final
644	C:\code-analysis\sonar-codes\Neuron.java	5	Public method and constructor comments are required

645	C:\code-analysis\sonar-codes\Neuron.java	17	Consider using varargs for methods or constructors which take an array the last parameter.
646	C:\code-analysis\sonar-codes\Neuron.java	17	Parameter 'inputs' is not assigned and could be declared final
647	C:\code-analysis\sonar-codes\Neuron.java	17	Public method and constructor comments are required
648	C:\code-analysis\sonar-codes\Neuron.java	20	Useless parentheses.
649	C:\code-analysis\sonar-codes\Neuron.java	20	Useless parentheses.
650	C:\code-analysis\sonar-codes\Neuron.java	29	Public method and constructor comments are required
651	C:\code-analysis\sonar-codes\Neuron.java	31	Local variable 'expmlx' could be declared final
652	C:\code-analysis\sonar-codes\Neuron.java	35	Returning '_synapticWeights' may expose an internal array.
653	C:\code-analysis\sonar-codes\Neuron.java	36	Avoid variables with short names like i
654	C:\code-analysis\sonar-codes\Neuron.java	36	Parameter 'i' is not assigned and could be declared final
655	C:\code-analysis\sonar-codes\Neuron.java	36	Public method and constructor comments are required

656	C:\code-analysis\sonar-codes\Neuron.java	37	Avoid variables with short names like i
657	C:\code-analysis\sonar-codes\Neuron.java	37	Avoid variables with short names like v
658	C:\code-analysis\sonar-codes\Neuron.java	37	Parameter 'i' is not assigned and could be declared final
659	C:\code-analysis\sonar-codes\Neuron.java	37	Parameter 'v' is not assigned and could be declared final
660	C:\code-analysis\sonar-codes\Neuron.java	37	Public method and constructor comments are required
661	C:\code-analysis\sonar-codes\Neuron.java	40	Field comments are required
662	C:\code-analysis\sonar-codes\Neuron.java	40	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
663	C:\code-analysis\sonar-codes\Neuron.java	40	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_activation' is not final.
664	C:\code-analysis\sonar-codes\Neuron.java	41	Field comments are required
665	C:\code-analysis\sonar-codes\Neuron.java	41	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
666	C:\code-analysis\sonar-codes\Neuron.java	41	Only variables that are final should contain underscores (except for underscores in standard prefix/suffix), '_synapticWeights' is not final.

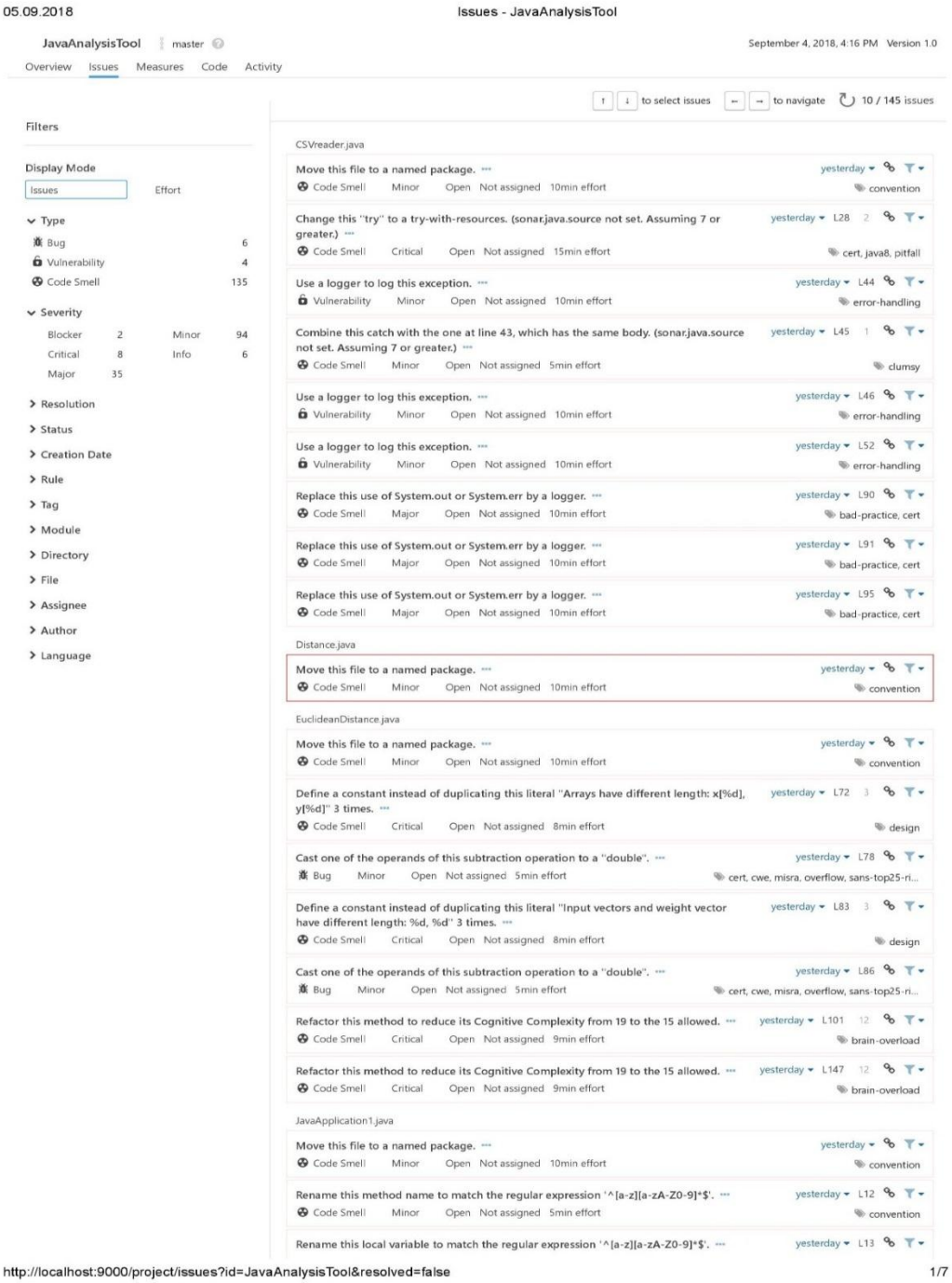
667	C:\code-analysis\sonar-codes\Neuron.java	44	Field comments are required
668	C:\code-analysis\sonar-codes\Neuron.java	44	Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes.
669	C:\code-analysis\sonar-codes\Neuron.java	44	To avoid mistakes add a comment at the beginning of the lambda field if you want a default access modifier
670	C:\code-analysis\sonar-codes\Neuron.java	44	Use explicit scoping instead of the default package private level
671	C:\code-analysis\sonar-codes\Neuron.java	44	Variables that are final and static should be all capitals, 'lambda' is not all capitals.
672	C:\code-analysis\sonar-codes\Server.java	12	All classes and interfaces must belong to a named package
673	C:\code-analysis\sonar-codes\Server.java	12	All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
674	C:\code-analysis\sonar-codes\Server.java	12	Header comments are required
675	C:\code-analysis\sonar-codes\Server.java	12	The utility class name 'Server' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper)'
676	C:\code-analysis\sonar-codes\Server.java	14	A method/constructor should not explicitly throw java.lang.Exception

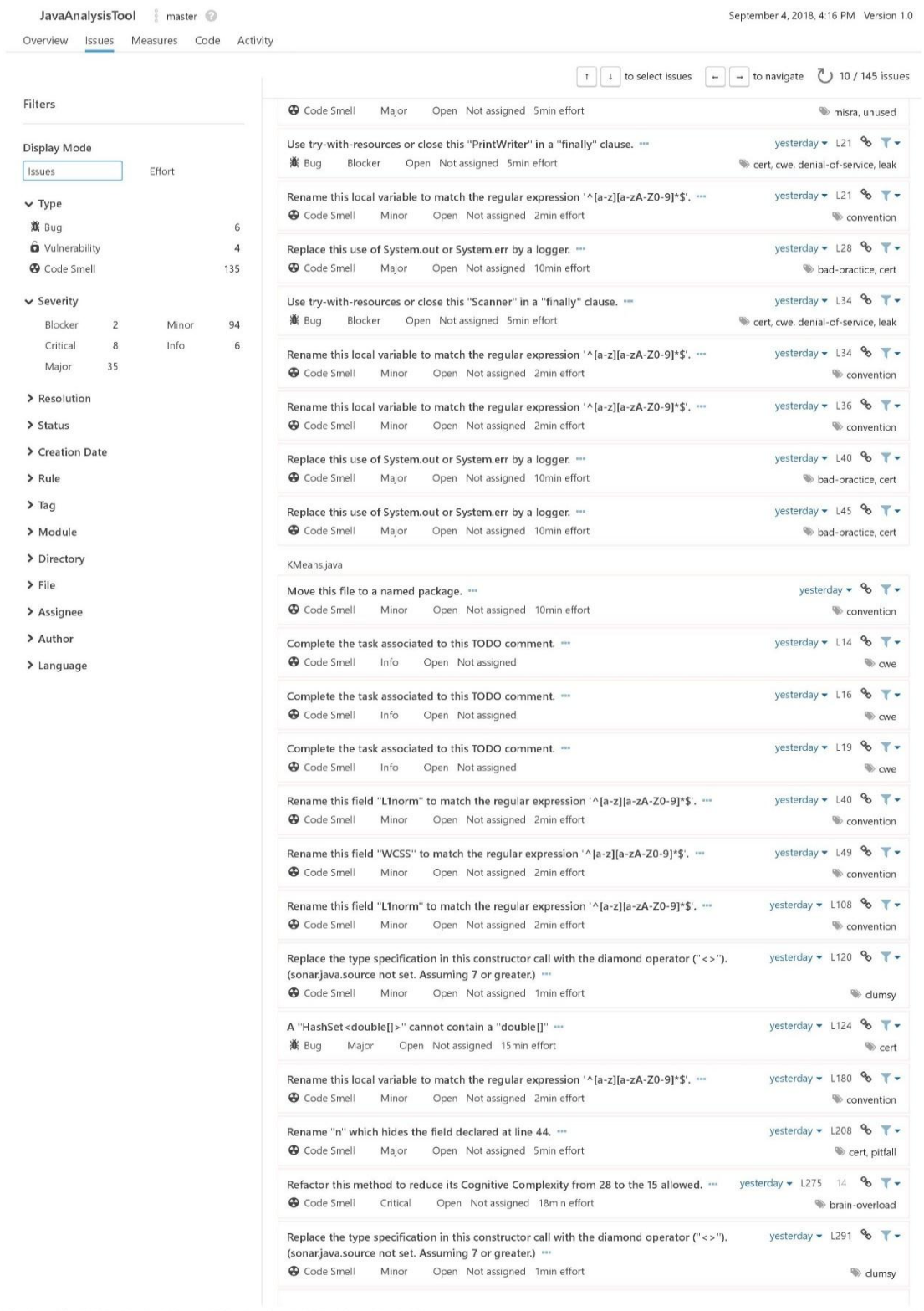
677	C:\code-analysis\sonar-codes\Server.java	14	Parameter 'args' is not assigned and could be declared final
678	C:\code-analysis\sonar-codes\Server.java	14	Public method and constructor comments are required
679	C:\code-analysis\sonar-codes\Server.java	15	Local variable 'port' could be declared final
680	C:\code-analysis\sonar-codes\Server.java	15	Potential violation of Law of Demeter (method chain calls)
681	C:\code-analysis\sonar-codes\Server.java	16	Local variable 'server' could be declared final
682	C:\code-analysis\sonar-codes\Server.java	23	All classes and interfaces must belong to a named package
683	C:\code-analysis\sonar-codes\Server.java	23	Header comments are required
684	C:\code-analysis\sonar-codes\Server.java	23	To avoid mistakes add a comment at the beginning of the RootHandler nested class if you want a default access modifier
685	C:\code-analysis\sonar-codes\Server.java	25	Avoid variables with short names like t
686	C:\code-analysis\sonar-codes\Server.java	25	Parameter 't' is not assigned and could be declared final
687	C:\code-analysis\sonar-codes\Server.java	26	Potential violation of Law of Demeter (method chain calls)

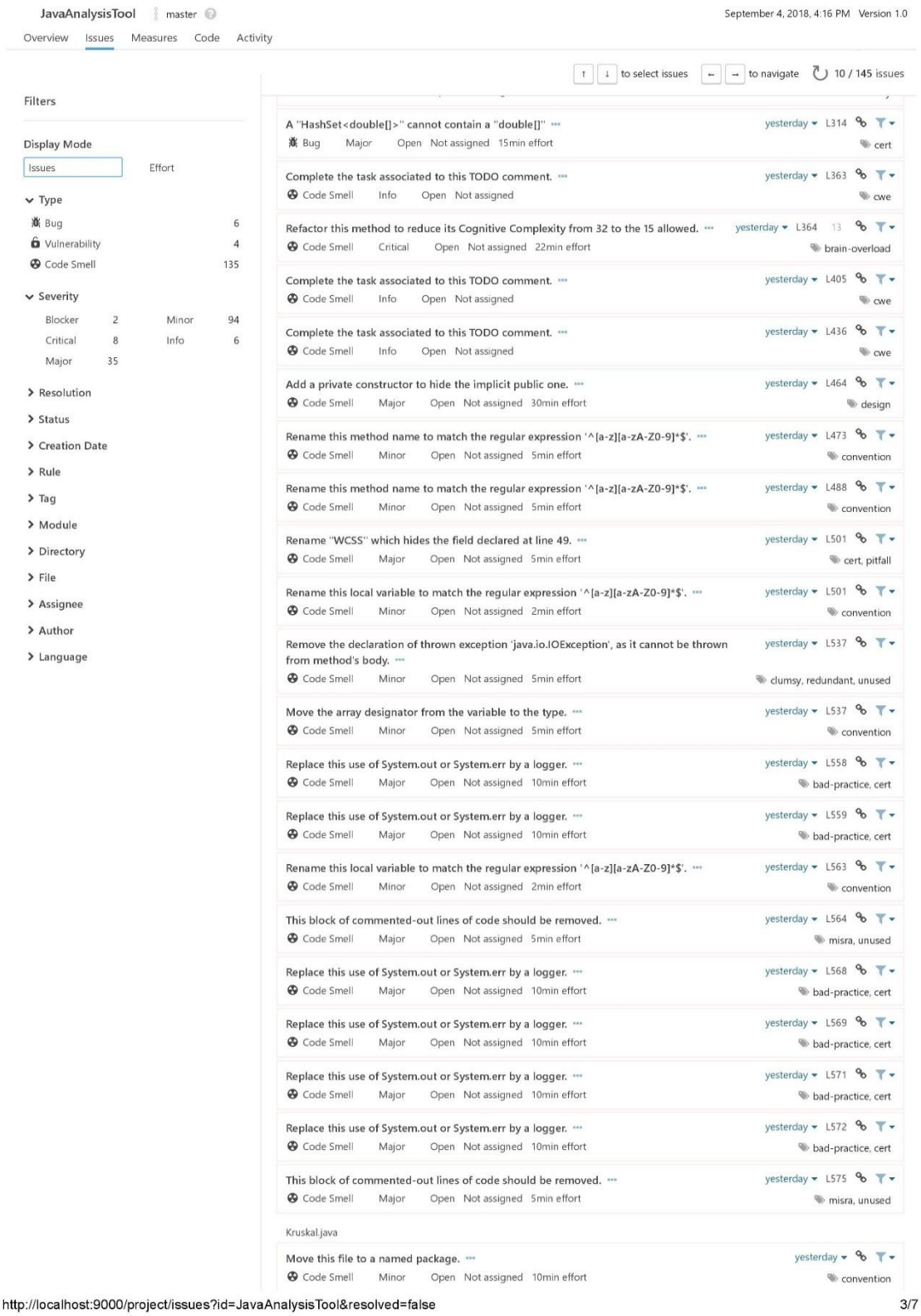
688	C:\code-analysis\sonar-codes\Server.java	26	Potential violation of Law of Demeter (method chain calls)
689	C:\code-analysis\sonar-codes\Server.java	27	Position literals first in String comparisons
690	C:\code-analysis\sonar-codes\Server.java	27	Potential violation of Law of Demeter (object not created locally)
691	C:\code-analysis\sonar-codes\Server.java	31	Local variable 'response' could be declared final
692	C:\code-analysis\sonar-codes\Server.java	32	Potential violation of Law of Demeter (object not created locally)
693	C:\code-analysis\sonar-codes\Server.java	33	Avoid variables with short names like os
694	C:\code-analysis\sonar-codes\Server.java	33	Local variable 'os' could be declared final
695	C:\code-analysis\sonar-codes\Server.java	34	Potential violation of Law of Demeter (object not created locally)
696	C:\code-analysis\sonar-codes\Server.java	34	Potential violation of Law of Demeter (object not created locally)
697	C:\code-analysis\sonar-codes\Server.java	35	Potential violation of Law of Demeter (object not created locally)
698	C:\code-analysis\sonar-codes\Server.java	39	Parameter 'path' is not assigned and could be declared final

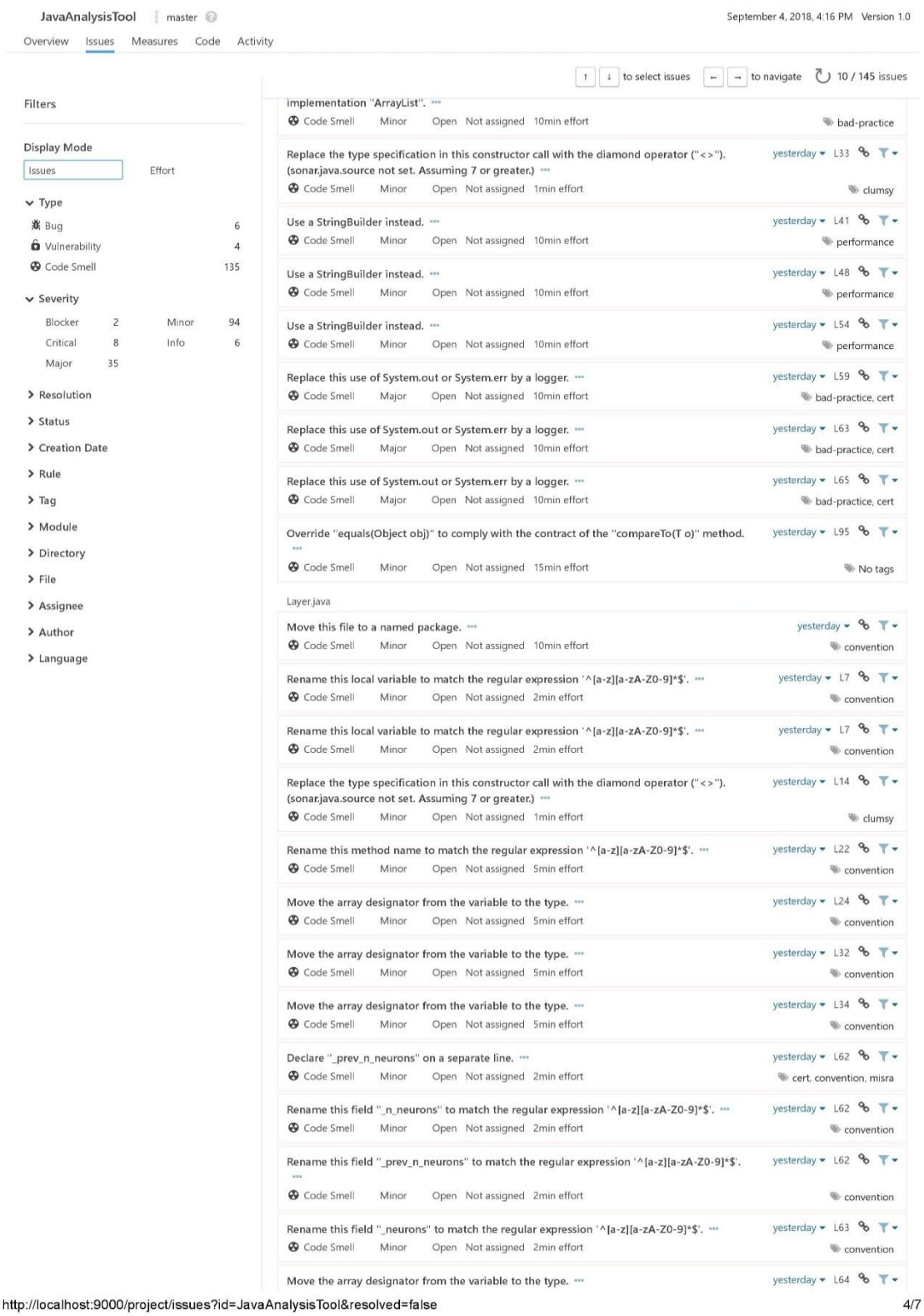
699	C:\code-analysis\sonar-codes\Server.java	39	To avoid mistakes add a comment at the beginning of the readContent method if you want a default access modifier
700	C:\code-analysis\sonar-codes\Server.java	39	Use explicit scoping instead of the default package private level
701	C:\code-analysis\sonar-codes\Server.java	41	Found 'DU'-anomaly for variable 'content' (lines '41'-'51').
702	C:\code-analysis\sonar-codes\Server.java	43	Avoid variables with short names like in
703	C:\code-analysis\sonar-codes\Server.java	43	Potential violation of Law of Demeter (method chain calls)
704	C:\code-analysis\sonar-codes\Server.java	44	Avoid assignments in operands
705	C:\code-analysis\sonar-codes\Server.java	44	Found 'DU'-anomaly for variable 'read' (lines '44'-'51').

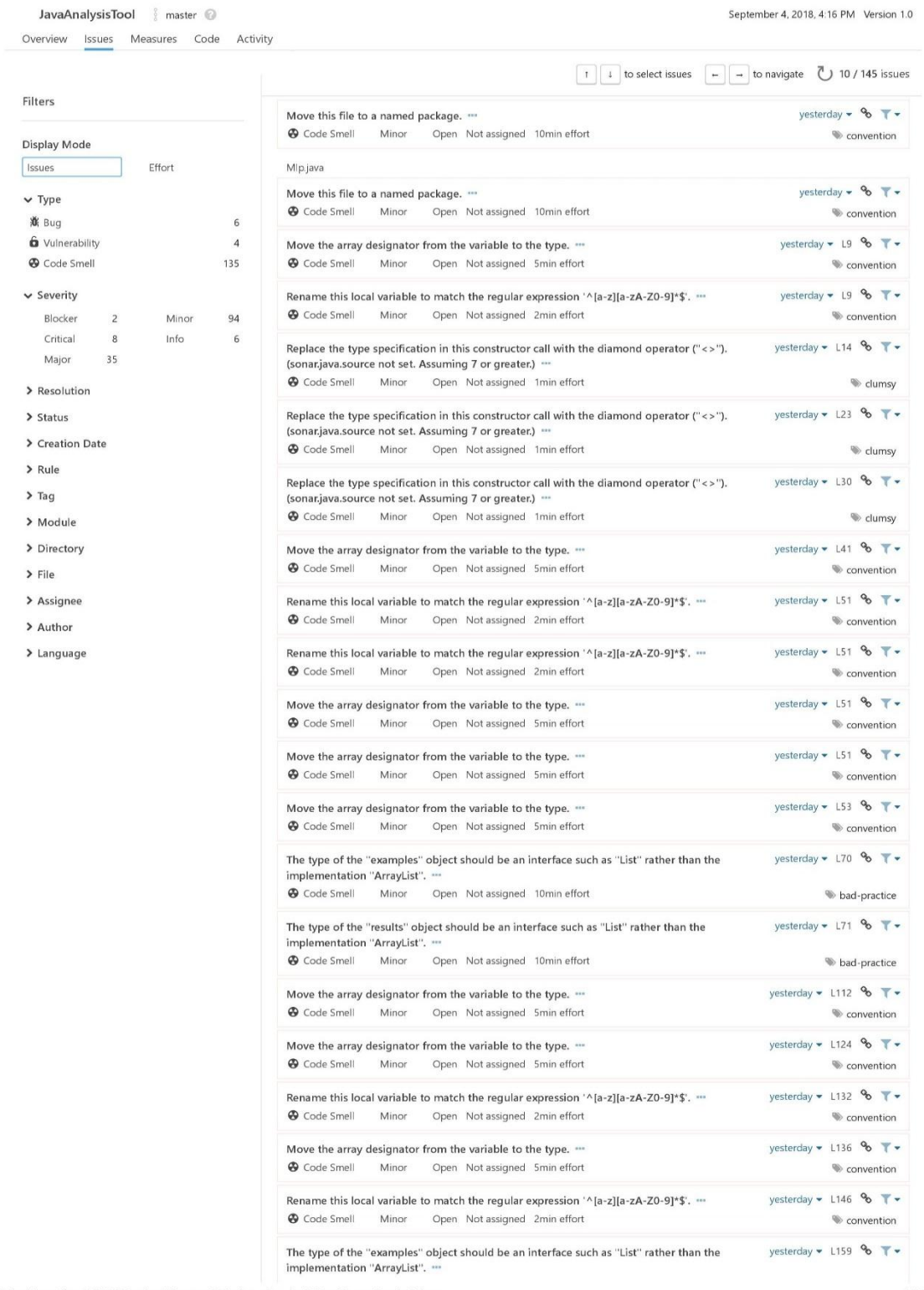
B
SonarQube Code Results

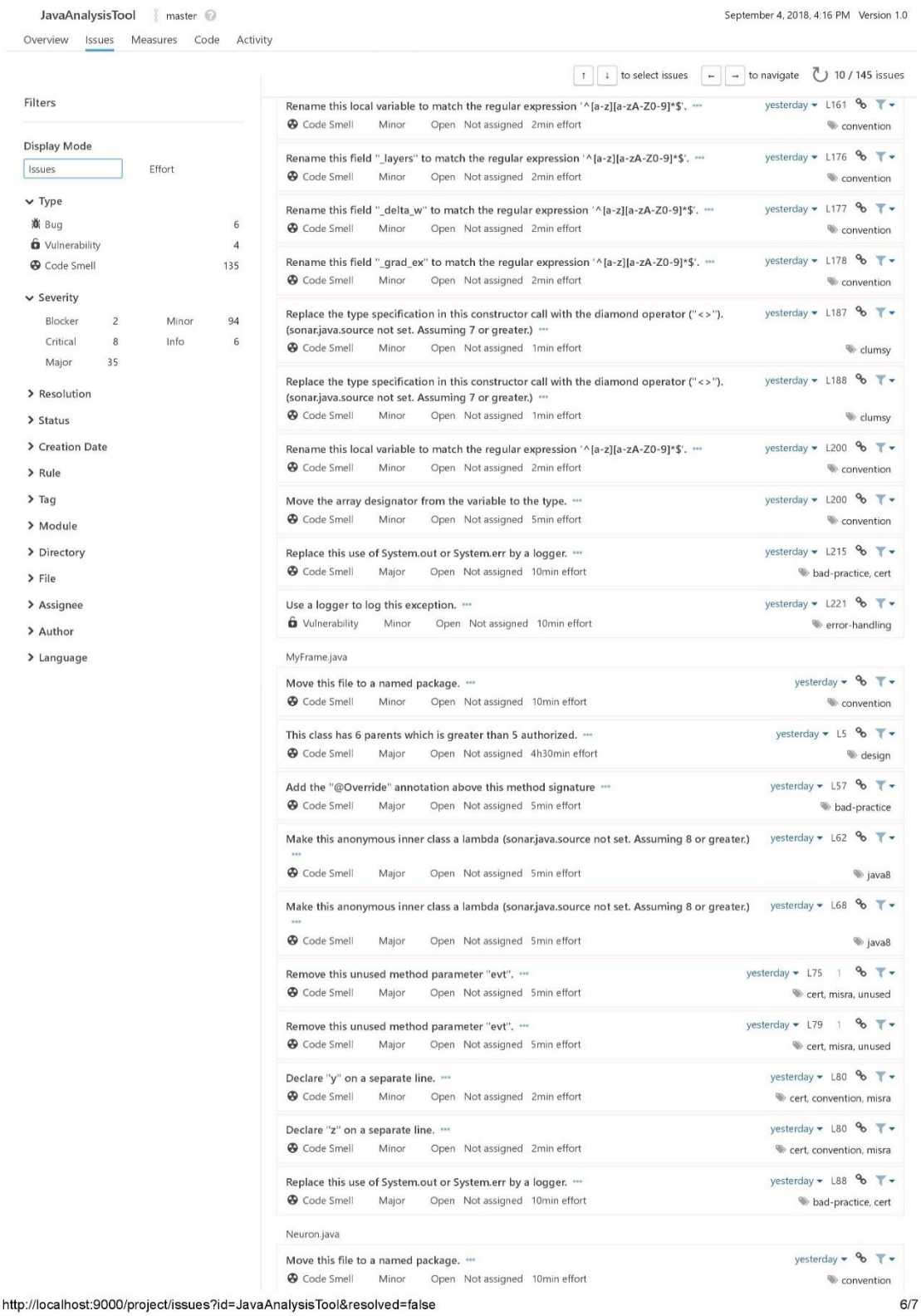


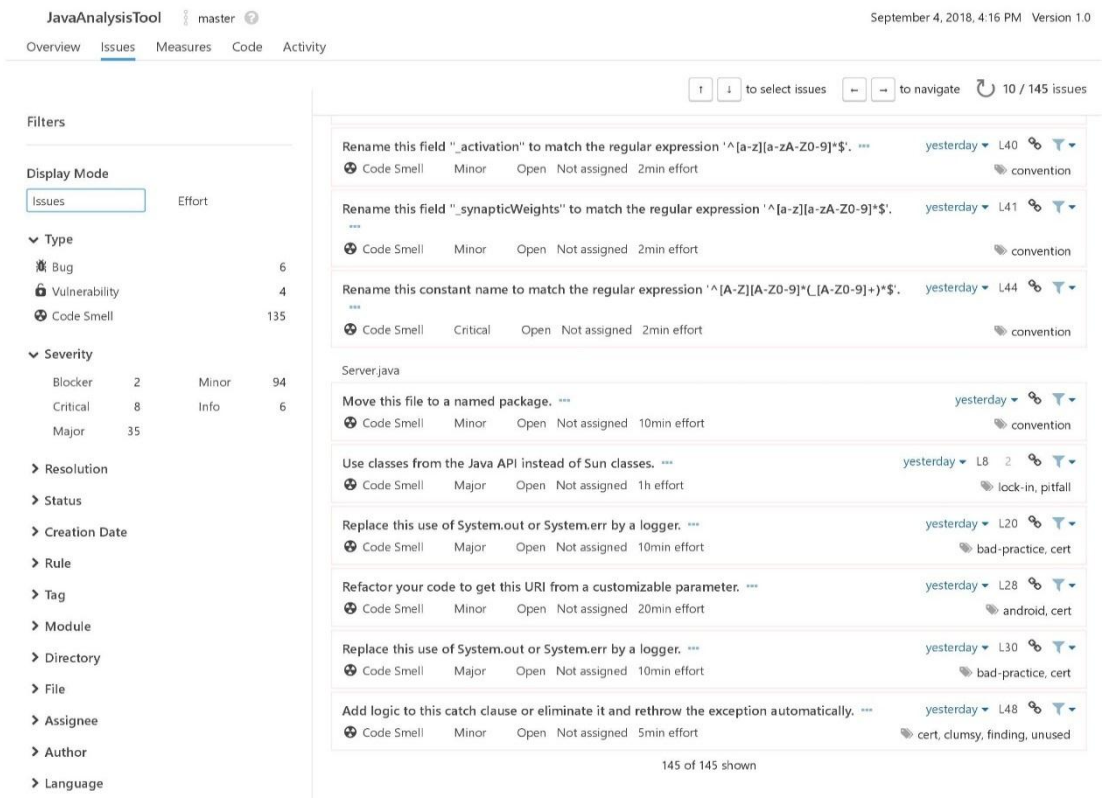

JavaAnalysisTool master
September 4, 2018, 4:16 PM Version 1.0
Overview Issues Measures Code Activity
to select issues to navigate 10 / 145 issues
Filters
Display Mode
Issues Effort
Type
Bug 6
Vulnerability 4
Code Smell 135
Severity
Blocker 2 Minor 94
Critical 8 Info 6
Major 35
Resolution
Status
Creation Date
Rule
Tag
Module
Directory
File
Assignee
Author
Language
Code Smell Major Open Not assigned 5min effort misra, unused
Use try-with-resources or close this "PrintWriter" in a "finally" clause. yesterday L21
Bug Blocker Open Not assigned 5min effort cert, cwe, denial-of-service, leak
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L21
Code Smell Minor Open Not assigned 2min effort convention
Replace this use of System.out or System.err by a logger. yesterday L28
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Use try-with-resources or close this "Scanner" in a "finally" clause. yesterday L34
Bug Blocker Open Not assigned 5min effort cert, cwe, denial-of-service, leak
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L34
Code Smell Minor Open Not assigned 2min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L36
Code Smell Minor Open Not assigned 2min effort convention
Replace this use of System.out or System.err by a logger. yesterday L40
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L45
Code Smell Major Open Not assigned 10min effort bad-practice, cert
KMeans.java
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention
Complete the task associated to this TODO comment. yesterday L14
Code Smell Info Open Not assigned cwe
Complete the task associated to this TODO comment. yesterday L16
Code Smell Info Open Not assigned cwe
Complete the task associated to this TODO comment. yesterday L19
Code Smell Info Open Not assigned cwe
Rename this field "L1norm" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L40
Code Smell Minor Open Not assigned 2min effort convention
Rename this field "WCSS" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L49
Code Smell Minor Open Not assigned 2min effort convention
Rename this field "L1norm" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L108
Code Smell Minor Open Not assigned 2min effort convention
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L120
Code Smell Minor Open Not assigned 1min effort clumsy
A "HashSet<double[]>" cannot contain a "double[]" yesterday L124
Bug Major Open Not assigned 15min effort cert
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L180
Code Smell Minor Open Not assigned 2min effort convention
Rename "n" which hides the field declared at line 44. yesterday L208
Code Smell Major Open Not assigned 5min effort cert, pitfall
Refactor this method to reduce its Cognitive Complexity from 28 to the 15 allowed. yesterday L275 14
Code Smell Critical Open Not assigned 18min effort brain-overload
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L291
Code Smell Minor Open Not assigned 1min effort clumsy


JavaAnalysisTool master
September 4, 2018, 4:16 PM Version 1.0
Overview Issues Measures Code Activity
to select issues to navigate 10 / 145 issues
Filters
Display Mode
Issues Effort
Type
Bug 6
Vulnerability 4
Code Smell 135
Severity
Blocker 2 Minor 94
Critical 8 Info 6
Major 35
Resolution
Status
Creation Date
Rule
Tag
Module
Directory
File
Assignee
Author
Language
A "HashSet<double[]>" cannot contain a "double[]" yesterday L314
Bug Major Open Not assigned 15min effort cert
Complete the task associated to this TODO comment. yesterday L363
Code Smell Info Open Not assigned cwe
Refactor this method to reduce its Cognitive Complexity from 32 to the 15 allowed. yesterday L364 13
Code Smell Critical Open Not assigned 22min effort brain-overload
Complete the task associated to this TODO comment. yesterday L405
Code Smell Info Open Not assigned cwe
Complete the task associated to this TODO comment. yesterday L436
Code Smell Info Open Not assigned cwe
Add a private constructor to hide the implicit public one. yesterday L464
Code Smell Major Open Not assigned 30min effort design
Rename this method name to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L473
Code Smell Minor Open Not assigned 5min effort convention
Rename this method name to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L488
Code Smell Minor Open Not assigned 5min effort convention
Rename "WCSS" which hides the field declared at line 49. yesterday L501
Code Smell Major Open Not assigned 5min effort cert, pitfall
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L501
Code Smell Minor Open Not assigned 2min effort convention
Remove the declaration of thrown exception 'java.io.IOException', as it cannot be thrown from method's body. yesterday L537
Code Smell Minor Open Not assigned 5min effort clumsy, redundant, unused
Move the array designator from the variable to the type. yesterday L537
Code Smell Minor Open Not assigned 5min effort convention
Replace this use of System.out or System.err by a logger. yesterday L558
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L559
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L563
Code Smell Minor Open Not assigned 2min effort convention
This block of commented-out lines of code should be removed. yesterday L564
Code Smell Major Open Not assigned 5min effort misra, unused
Replace this use of System.out or System.err by a logger. yesterday L568
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L569
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L571
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L572
Code Smell Major Open Not assigned 10min effort bad-practice, cert
This block of commented-out lines of code should be removed. yesterday L575
Code Smell Major Open Not assigned 5min effort misra, unused
Kruskal.java
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention


JavaAnalysisTool master
September 4, 2018, 4:16 PM Version 1.0
Overview Issues Measures Code Activity
to select issues to navigate 10 / 145 issues
Filters
Display Mode
Issues Effort
Type
Bug 6
Vulnerability 4
Code Smell 135
Severity
Blocker 2 Minor 94
Critical 8 Info 6
Major 35
Resolution
Status
Creation Date
Rule
Tag
Module
Directory
File
Assignee
Author
Language
implementation "ArrayList". Code Smell Minor Open Not assigned 10min effort bad-practice
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L33 Code Smell Minor Open Not assigned 1min effort clumsy
Use a StringBuilder instead. yesterday L41 Code Smell Minor Open Not assigned 10min effort performance
Use a StringBuilder instead. yesterday L48 Code Smell Minor Open Not assigned 10min effort performance
Use a StringBuilder instead. yesterday L54 Code Smell Minor Open Not assigned 10min effort performance
Replace this use of System.out or System.err by a logger. yesterday L59 Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L63 Code Smell Major Open Not assigned 10min effort bad-practice, cert
Replace this use of System.out or System.err by a logger. yesterday L65 Code Smell Major Open Not assigned 10min effort bad-practice, cert
Override "equals(Object obj)" to comply with the contract of the "compareTo(T o)" method. yesterday L95 Code Smell Minor Open Not assigned 15min effort No tags
Layer.java
Move this file to a named package. yesterday Code Smell Minor Open Not assigned 10min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L7 Code Smell Minor Open Not assigned 2min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L7 Code Smell Minor Open Not assigned 2min effort convention
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L14 Code Smell Minor Open Not assigned 1min effort clumsy
Rename this method name to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L22 Code Smell Minor Open Not assigned 5min effort convention
Move the array designator from the variable to the type. yesterday L24 Code Smell Minor Open Not assigned 5min effort convention
Move the array designator from the variable to the type. yesterday L32 Code Smell Minor Open Not assigned 5min effort convention
Move the array designator from the variable to the type. yesterday L34 Code Smell Minor Open Not assigned 5min effort convention
Declare "_prev_n_neurons" on a separate line. yesterday L62 Code Smell Minor Open Not assigned 2min effort cert, convention, misra
Rename this field "_n_neurons" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L62 Code Smell Minor Open Not assigned 2min effort convention
Rename this field "_prev_n_neurons" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L62 Code Smell Minor Open Not assigned 2min effort convention
Rename this field "_neurons" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L63 Code Smell Minor Open Not assigned 2min effort convention
Move the array designator from the variable to the type. yesterday L64


JavaAnalysisTool master
September 4, 2018, 4:16 PM Version 1.0
Overview Issues Measures Code Activity
to select issues to navigate 10 / 145 issues
Filters
Display Mode
Issues Effort
Type
Bug 6
Vulnerability 4
Code Smell 135
Severity
Blocker 2 Minor 94
Critical 8 Info 6
Major 35
Resolution
Status
Creation Date
Rule
Tag
Module
Directory
File
Assignee
Author
Language
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention
MIp.java
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention
Move the array designator from the variable to the type. yesterday L9
Code Smell Minor Open Not assigned 5min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L9
Code Smell Minor Open Not assigned 2min effort convention
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L14
Code Smell Minor Open Not assigned 1min effort clumsy
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L23
Code Smell Minor Open Not assigned 1min effort clumsy
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L30
Code Smell Minor Open Not assigned 1min effort clumsy
Move the array designator from the variable to the type. yesterday L41
Code Smell Minor Open Not assigned 5min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L51
Code Smell Minor Open Not assigned 2min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L51
Code Smell Minor Open Not assigned 2min effort convention
Move the array designator from the variable to the type. yesterday L51
Code Smell Minor Open Not assigned 5min effort convention
Move the array designator from the variable to the type. yesterday L51
Code Smell Minor Open Not assigned 5min effort convention
Move the array designator from the variable to the type. yesterday L53
Code Smell Minor Open Not assigned 5min effort convention
The type of the "examples" object should be an interface such as "List" rather than the implementation "ArrayList". yesterday L70
Code Smell Minor Open Not assigned 10min effort bad-practice
The type of the "results" object should be an interface such as "List" rather than the implementation "ArrayList". yesterday L71
Code Smell Minor Open Not assigned 10min effort bad-practice
Move the array designator from the variable to the type. yesterday L112
Code Smell Minor Open Not assigned 5min effort convention
Move the array designator from the variable to the type. yesterday L124
Code Smell Minor Open Not assigned 5min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L132
Code Smell Minor Open Not assigned 2min effort convention
Move the array designator from the variable to the type. yesterday L136
Code Smell Minor Open Not assigned 5min effort convention
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L146
Code Smell Minor Open Not assigned 2min effort convention
The type of the "examples" object should be an interface such as "List" rather than the implementation "ArrayList". yesterday L159


JavaAnalysisTool master
September 4, 2018, 4:16 PM Version 1.0
Overview Issues Measures Code Activity
to select issues to navigate 10 / 145 issues
Filters
Display Mode
Issues Effort
Type
Bug 6
Vulnerability 4
Code Smell 135
Severity
Blocker 2 Minor 94
Critical 8 Info 6
Major 35
Resolution
Status
Creation Date
Rule
Tag
Module
Directory
File
Assignee
Author
Language
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L161
Code Smell Minor Open Not assigned 2min effort convention
Rename this field "_layers" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L176
Code Smell Minor Open Not assigned 2min effort convention
Rename this field "_delta_w" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L177
Code Smell Minor Open Not assigned 2min effort convention
Rename this field "_grad_ex" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L178
Code Smell Minor Open Not assigned 2min effort convention
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L187
Code Smell Minor Open Not assigned 1min effort clumsy
Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) yesterday L188
Code Smell Minor Open Not assigned 1min effort clumsy
Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L200
Code Smell Minor Open Not assigned 2min effort convention
Move the array designator from the variable to the type. yesterday L200
Code Smell Minor Open Not assigned 5min effort convention
Replace this use of System.out or System.err by a logger. yesterday L215
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Use a logger to log this exception. yesterday L221
Vulnerability Minor Open Not assigned 10min effort error-handling
MyFrame.java
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention
This class has 6 parents which is greater than 5 authorized. yesterday L5
Code Smell Major Open Not assigned 4h30min effort design
Add the "@Override" annotation above this method signature yesterday L57
Code Smell Major Open Not assigned 5min effort bad-practice
Make this anonymous inner class a lambda (sonar.java.source not set. Assuming 8 or greater.) yesterday L62
Code Smell Major Open Not assigned 5min effort java8
Make this anonymous inner class a lambda (sonar.java.source not set. Assuming 8 or greater.) yesterday L68
Code Smell Major Open Not assigned 5min effort java8
Remove this unused method parameter "evt". yesterday L75 1
Code Smell Major Open Not assigned 5min effort cert, misra, unused
Remove this unused method parameter "evt". yesterday L79 1
Code Smell Major Open Not assigned 5min effort cert, misra, unused
Declare "y" on a separate line. yesterday L80
Code Smell Minor Open Not assigned 2min effort cert, convention, misra
Declare "z" on a separate line. yesterday L80
Code Smell Minor Open Not assigned 2min effort cert, convention, misra
Replace this use of System.out or System.err by a logger. yesterday L88
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Neuron.java
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention


JavaAnalysisTool master
September 4, 2018, 4:16 PM Version 1.0
Overview Issues Measures Code Activity
to select issues to navigate 10 / 145 issues
Filters
Display Mode
Issues Effort
Type
Bug 6
Vulnerability 4
Code Smell 135
Severity
Blocker 2 Minor 94
Critical 8 Info 6
Major 35
Resolution
Status
Creation Date
Rule
Tag
Module
Directory
File
Assignee
Author
Language
Rename this field "_activation" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L40
Code Smell Minor Open Not assigned 2min effort convention
Rename this field "_synapticWeights" to match the regular expression '^[a-z][a-zA-Z0-9]*$'. yesterday L41
Code Smell Minor Open Not assigned 2min effort convention
Rename this constant name to match the regular expression '^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$'. yesterday L44
Code Smell Critical Open Not assigned 2min effort convention
Server.java
Move this file to a named package. yesterday
Code Smell Minor Open Not assigned 10min effort convention
Use classes from the Java API instead of Sun classes. yesterday L8
Code Smell Major Open Not assigned 1h effort lock-in, pitfall
Replace this use of System.out or System.err by a logger. yesterday L20
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Refactor your code to get this URI from a customizable parameter. yesterday L28
Code Smell Minor Open Not assigned 20min effort android, cert
Replace this use of System.out or System.err by a logger. yesterday L30
Code Smell Major Open Not assigned 10min effort bad-practice, cert
Add logic to this catch clause or eliminate it and rethrow the exception automatically. yesterday L48
Code Smell Minor Open Not assigned 5min effort cert, clumsy, finding, unused
145 of 145 shown

REFERENCES

[1] N. Bhateja, "A Study on Various Software Automation Testing Tools", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, no. 6, 2015.

[2] S. Sharma, "Study And Analysis Of Automation Testing Techniques", Journal of Global Research in Computer Science, vol. 3, no. 12, 2012.

[3] "Analysis of Automation and Manual Testing Using Software Testing Tool", IJIACS, vol. 4, ISSN 2347 – 8616, 2017.

[4] Narayan, H. Black, R. "Implementing a Metrics Program to Guide Quality Improvements", Testing Experience 11, 2010.

[5] Premal, B. Nirpal et al, "A Brief of Software Testing Metrics, International Journal on Computer Science and Engineering", Vol. 3, No 1, 2011.

[6] Singh, Y., Malhotra, R., "What Should We Measure During Testing?", Testing Experience, 11, 52-54, 2010.

[7] Black, R., Mitchell, J. L., "Advanced Software Testing – Vol. 3: Guide To The ISTQB® Advanced Certification as an Advanced Technical Test Analyst", Rocky Nook, 2011.

[8] Ebert C., Dumke R., Bundschuh M., Schmietendorf A., "Best Practices in Software Measurement", Berlin Heidelberg, Springer, 2005.

[9] Erdoğmuş H., "Tracking Progress through Earned Value", IEEE Software, 27. 2-7. 10.1109/MS, 2010.

[10] Valerdi, Dr. R. Peña, M., "Practical Software and Systems Measurement: A Foundation for Objective Project Management", v. 4.0b1, PSM Guide.4.0b.Part 5, 2010.

[11] Watts H. S., "Managing the Software Process", Massachusetts, MA, Addison-Wesley, 1990.

[12] P. Kulik, "A practical approach to software metrics," in *IT Professional*, vol. 2, no. 1, pp. 38-42, Jan.-Feb. 2000.

[13] Software Development Cost Estimating Guidebook, Software Technology Support Center Cost Analysis Group, July 2009.

[14] R. W. Selby, "Enabling reuse-based software development of large-scale systems," in IEEE Transactions on Software Engineering, vol. 31, no. 6, pp. 495-510, June 2005.

[15] E. Arisholm, L. C. Briand and A. Foyen, "Dynamic coupling measurement for object-oriented software," in IEEE Transactions on Software Engineering, vol. 30, no. 8, pp. 491-506, Aug. 2004.

[16] Briand, L.C., Daly, J.W. & Wüst, J., "Empirical Software Engineering", Pages 3: 6, 1998.

[17] Choi K.H.T., Tempero E., "Dynamic Measurement of Polymorphism", ACSC '07 Proceedings of the thirtieth Australasian conference on Computer science - Volume 62 Pages 211-220, 2007.

[18] K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra,: “Software Design Metrics for Object-Oriented Software”, in Journal of Object Technology, vol. 6. no. 1, January-February, pp. 121-138, 2006.

[19] Stephen H. Kan, Metrics and Models of Software Quality Engineering, 2nd Edition, Chapter 4.3 Metrics for Software Maintenance, Addison-Wesley Professional, 2002.

[20] Premal, B., Nirpal et al., "A Brief of Software Testing Metrics, International Journal on Computer Science and Engineering", Vol. 3, No 1, 2011.

[21] Hass, A, M, J., "Guide to Advanced Software Testing", Artech House, 2008.

[22] Singh, Y., Malhotra, R., "What Should We Measure During Testing? ", Testing Experience, 11, 52-54, 2010.

[23] Hutcheson, M., L., "Software Testing Fundamentals: Methods and Metrics", John Wiley&Sons, 2003.

[24] Freddy Mallet , "SONAR is becoming SONARQUBE". SonarQube project mailing list. Retrieved 3 July 2013.

[25] Campell/Papapetrou, Ann/Patroklos, "Sonar (SonarQube) in action", Greenwich, Connecticut, USA: Manning Publications. p. 350, 2013.

[26] Buijze, Allard (2010-02-26). "Measuring Code Quality with Sonar". Retrieved 29 August 2017.

[27] Hazrati, Vikas (2010-03-30). "Monetizing the Technical Debt". Retrieved 29 August 2017.

[28] P. Chandra, B. Chess and J. Steven, "Putting the tools to work: how to succeed with source code analysis," in *IEEE Security & Privacy*, vol. 4, no. 3, pp. 80-83, May-June 2006.

[29] Anderson, P., "The Use and Limitations of Static-Analysis Tools to Improve Software Quality", CrossTalk-Journal of Defense Software Engineering. 21, 2008.

[30] Sotirov, A. I., “Automatic Vulnerability Detection using Static Source Code Analysis”, The University of Alabama, 2005.

[31] P. Louridas, "Static code analysis," in *IEEE Software*, vol. 23, no. 4, pp. 58-61, July-Aug. 2006.

[32] German, A., “Software Static Code Analysis Lessons Learnt”, CrossTalk, November 2003.

[33] Baca, D., Carlsson, B. ve Lundberg, L., “Evaluating the Cost Reduction of Static Code Analysis for Software Security”, Proceedings of the third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS’08), 2008.

[34] Oskouei, E. H. Kalıpsız, O., “Comparing Bug Finding Tools for Java Open Source Software“, IMISC 2018 Conference Proceedings, 17-19, 2018.

Publications from the Thesis

Contact Information: elmira.ha2006@gmail.com

Conference Papers

1. Oskouei, E. H. Kalıpsız, O., "Comparing Bug Finding Tools for Java Open Source Software", IMISC 2018 Conference Proceedings, 17-19, 2018.