A Project Report on

Fault Prediction Model For Identifying Faulty Classes

BY

**Simran Sandhu**

**2017A7PS1454H**

Under the supervision of

**Dr. Lov Kumar**

**SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF**

**CS F376 Design Project**



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)**

**HYDERABAD CAMPUS**

**(December 2019**)

**ACKNOWLEDGMENTS**

I am extremely thankful to Dr. Lov Kumar, Assistant Professor, Bits-Pilani Hyderabad for guiding me at every step and keeping me motivated to complete the project.



**Birla Institute of Technology and Science-Pilani,**

**Hyderabad Campus**

**Certificate**

This is to certify that the project report entitled “**Fault Prediction Model For Identifying Faulty Classes”** submitted by Ms. Simran Sandhu (ID No.2017A7PS1454H) in partial fulfillment of the requirements of the course CS F376, Design Project Course, embodies the work done by her under my supervision and guidance.

**Date: December 30th,2019 Dr. Lov Kumar**

BITS- Pilani, Hyderabad Campus

**ABSTRACT**

For object-oriented applications, prediction models using design metrics can be used to identify faulty classes early on. If these faulty software components can be detected early in the development project's life cycle,mitigating actions can be taken, such as a redesign. In this project we study object-oriented design metrics to construct such prediction models. The study used data collected from tera-PROMISE Repository. We used a source code metrics validation method and found that only a few source code metrics were relevant in building a model.

**TABLE OF CONTENTS**

Title page……………………………………………………………….1

Acknowledgements…………………………………………………….2

Certificate……………………………………………………………....3

Abstract………………………………………………………………...4

1.Introduction....………………………………………….………….....6

3.Dataset....………………………………………………………..........7

4.Ranksum Test.…………………………………………………..........9

5.Pearson Correlation.…………………………………………………10

6.Weka Tool.…………………………………………………..............11

6.Methodology…………………………………………………….......13

6.Code……………………………………………………....................14

Results & Conclusion…………………………………………………16

References……………………………………….……………………33

**Introduction**

In the realm of object-oriented systems, one approach to identify faulty classes early in development is to construct prediction models using source code metrics. For creating a fault prediction model the first step is to analyse the source code metrics and finding which of them are relevant to us. Ranksum test and pearson’s correlation has been employed to determine which of the selected sets of source code metrics work better or whether they all perform equally well.

**Dataset**

The fault data of fifty six projects are taken from tera-PROMISE Repository[9]. The data set contains bug information and twenty software source code metric listed as follows:

1. Weighted Methods per Class **(WMC)**[1] :the sum of all the complexities of the methods in the class

2. Number of Children **(NOC)**[1] :the number of immediate subclasses of a class

3. Response For Class **(RFC)**[1] :set of methods that can potentially be executed in response to a message received by an object of that class.

4. Depth of Inheritance Tree **(DIT)**[1] :Depth of Inheritance Tree (DIT) is a count of the classes that a particular class inherits from.

5. Coupling between objects **(CBO)**[1]: is a count of the number of classes that are coupled to a particular class

6. Lack of cohesion in methods **(LCOM)**[1]: Number of sets of methods in a class that are not related through the sharing of some of the class's fields

7. Afferent coupling **(Ca)**[2]: Number of other classes use the specific class

8. Efferent coupling **(Ce)**[2]: Number of classes used by the specific class

9. Number of public methods **(NPM)**[3]: Number of methods in a class that are declared as public.

10. **LCOM3[**4]: Lack of cohesion in methods Henderson-Sellers version

11. Lines of code **(LOC)**[5]: Number of lines in the text of the source code

12. Data access metric **(DAM)**[3]: Ratio of the number of private (protected) attributes to the total number of attributes declared in the class

13. Measure of aggregation **(MOA)**[3] :Number of data declarations whose types are user defined classes

14. Measure of functional abstraction **(MFA)**[3]: Ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class.

15. Cohesion among methods of class **(CAM)**[3]: Sum of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.

16. Inheritance coupling **(IC)**[6]: Number of parent classes to which a given class is coupled.

17. Coupling between methods **(CBM)**[6]: Number of new/redefined methods to which all the inherited methods are coupled

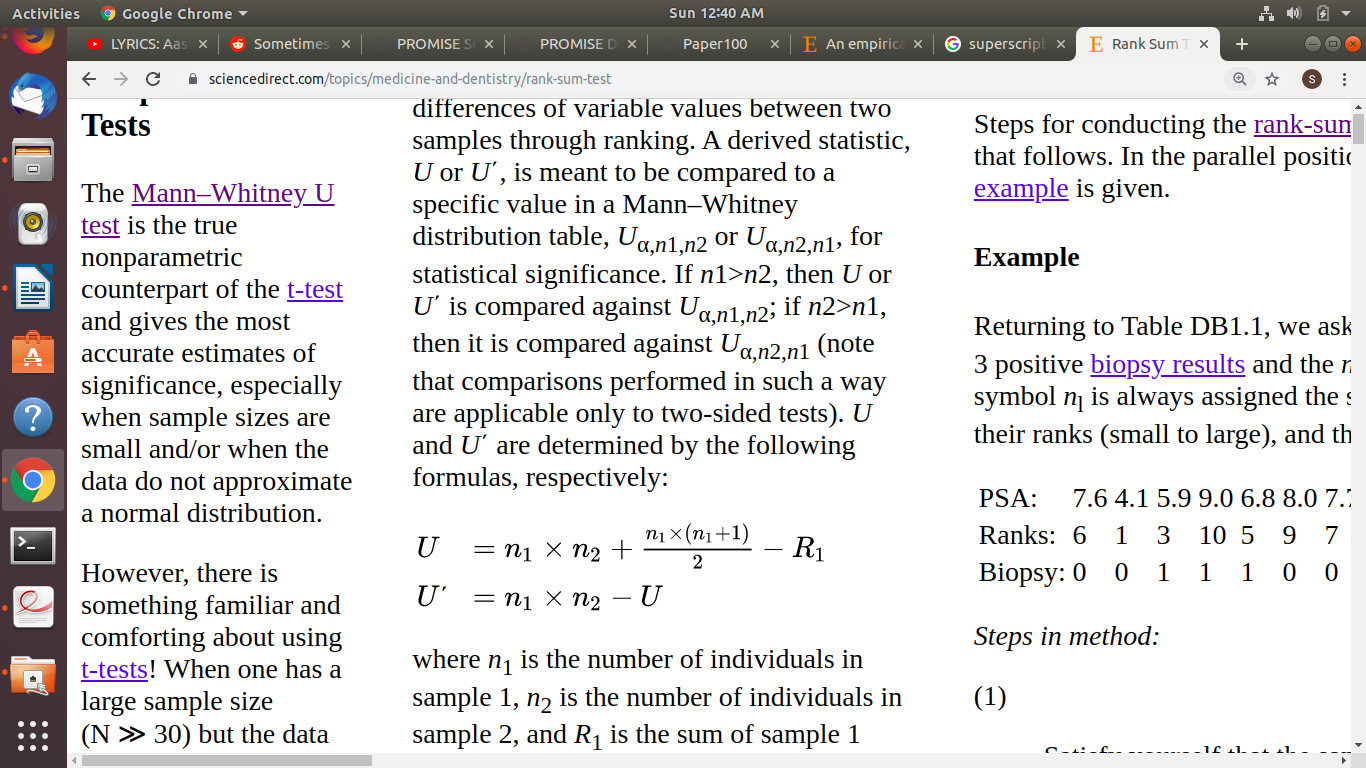
18. Average method complexity **(AMC)**[6]: Average method size for each class.

19. Max cyclomatic complexity **(Max − CC)**[7]: Maximum cyclomatic complexity of methods defined in a class

20. Average cyclomatic complexity **(Avg − CC)**[7]: Average cyclomatic complexity of methods defined in a class

**Ranksum Test**

The Ranksum test aims to detect differences of variable values between two samples through ranking. A derived statistic, U or U΄, is meant to be compared to a specific value in a Mann–Whitney distribution table, Uα,n1,n2 or Uα,n2,n1, for statistical significance. If n1>n2, then U or U΄ is compared against Uα,n1,n2; if n2>n1, then it is compared against Uα,n2,n1 (note that comparisons performed in such a way are applicable only to two-sided tests). U and U΄ are determined by the following formulas, respectively:

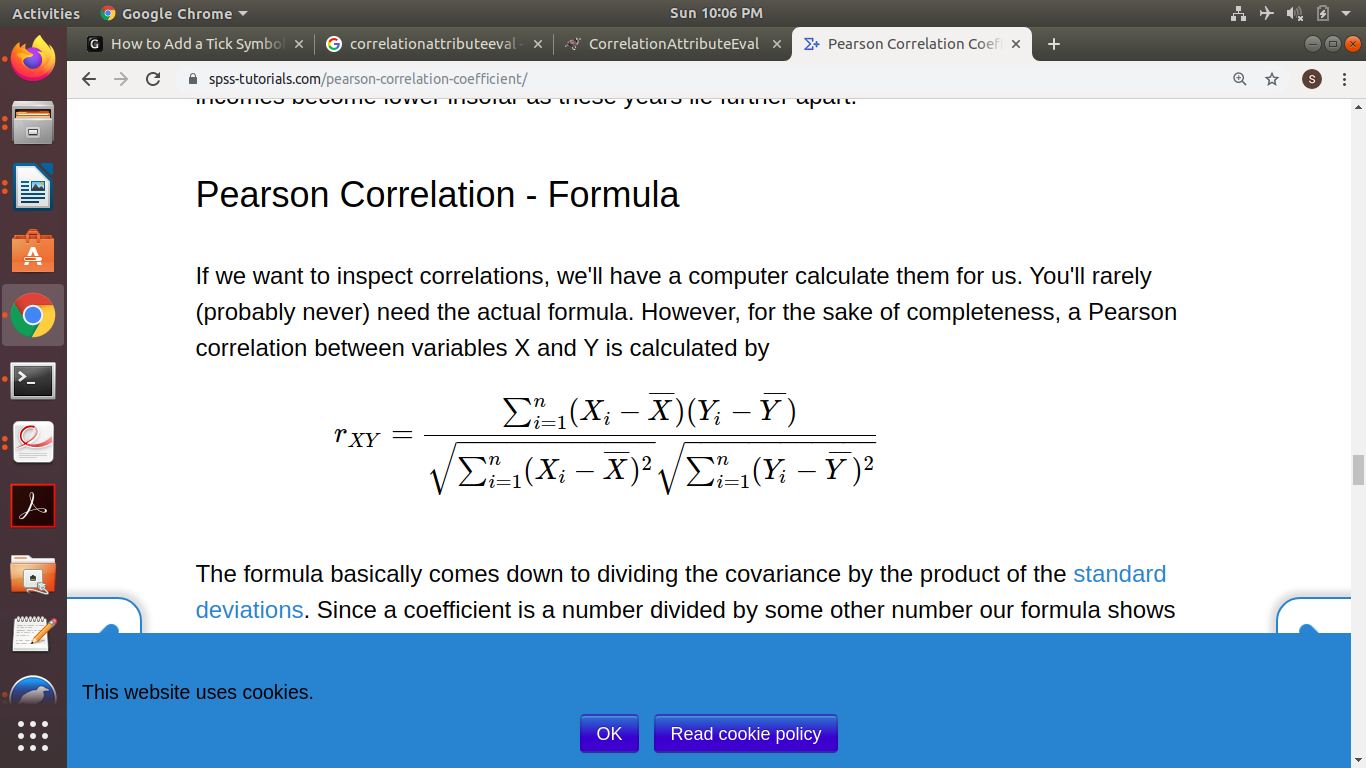


where n1 is the number of individuals in sample 1, n2 is the number of individuals in sample 2, and R1 is the sum of sample 1 ranks.[8]

**PEARSON CORRELATION**

A Pearson correlation is a number between -1 and 1 that indicates the extent to which two variables are linearly related.

Pearson correlation between variables X and Y is calculated by



The formula basically comes down to dividing the covariance by the product of the [standard deviations](https://www.spss-tutorials.com/standard-deviation-what-is-it/).

**WEKA TOOL**

Waikato Environment for Knowledge Analysis, developed at the University of Waikato, New Zealand. It is free software licensed under the GNU General Public License.Weka is a collection of machine learning algorithms for data mining tasks.Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

CorrelationAttributeEval Attribute Evaluator (which uses Pearson’s Correlation) with the search method Ranker is used for attribute selection. The 20 attributes were listed in descending order with respect to the value of their correlation with the bug class.

In that ranking top logN (i.e 5) features are selected for each project.



**LogN Feature Selection using Weka Tool**

**Methodology**

**A.Ranksum**

1. The csv files containing information about the projects are read into a TabularTextDatastore object using the tabularTextDatastore function.
2. A 56\*20 matrix was constructed using the ranksum function of the matlab library in which each column corresponds to a source code metric p value and each row corresponds to a project.
3. For a value of p<0.05 a value of p=0 was assigned and p=1 otherwise.
4. The resulting matrix is stored in an excel file for further analysis.

**B.Pearson’s Correlation**

1. For each project pearson’s correlation is calculated using corrcoef inbuilt function in MATLAB.
2. If a source code metric shows a high correlation (i.e a a value of r<=-0.7 or r>=0.7) we select this source code metric ( a value of 1 is given for selected metrics).
3. The resulting matrices are stored in an excel file, each sheet representing a separate project.

**Code**

**A.Ranksum**

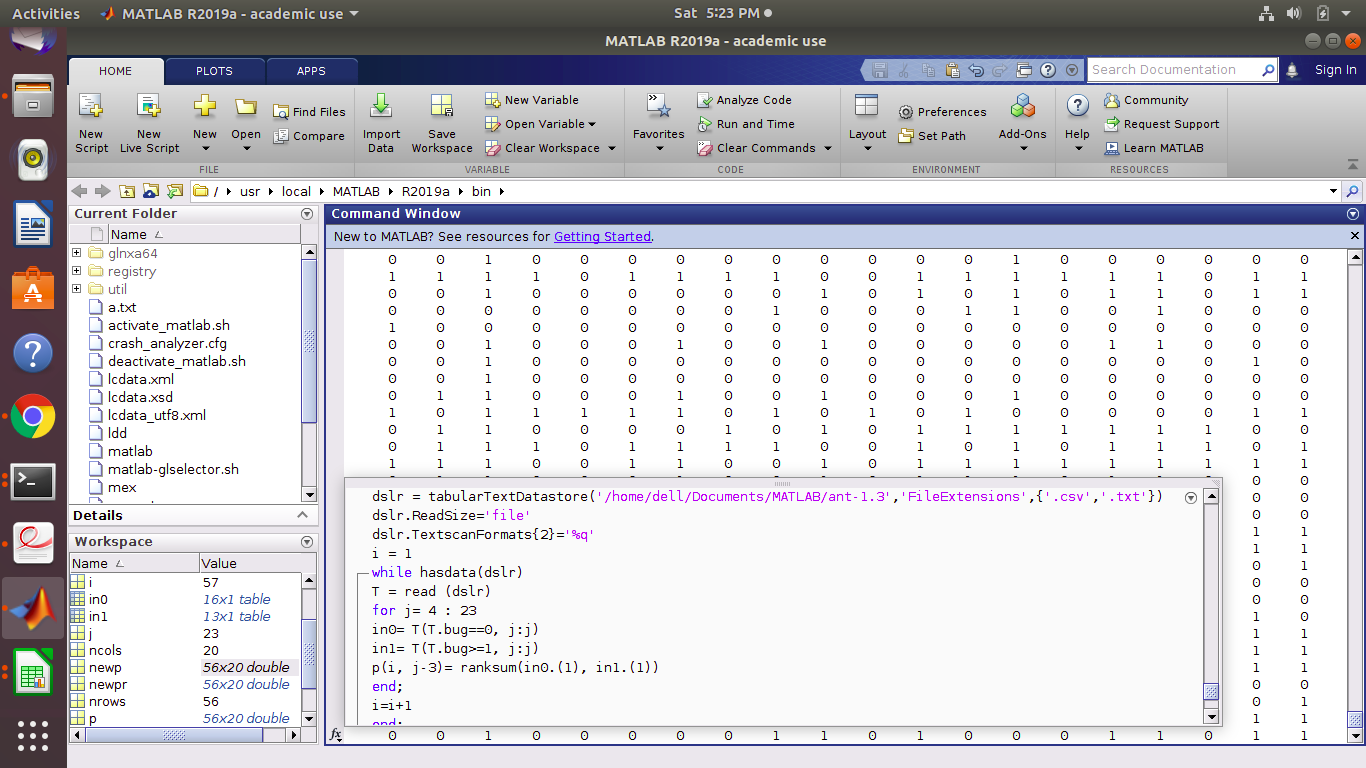


FIG 1: For calculating p value using ranksum matlab library function

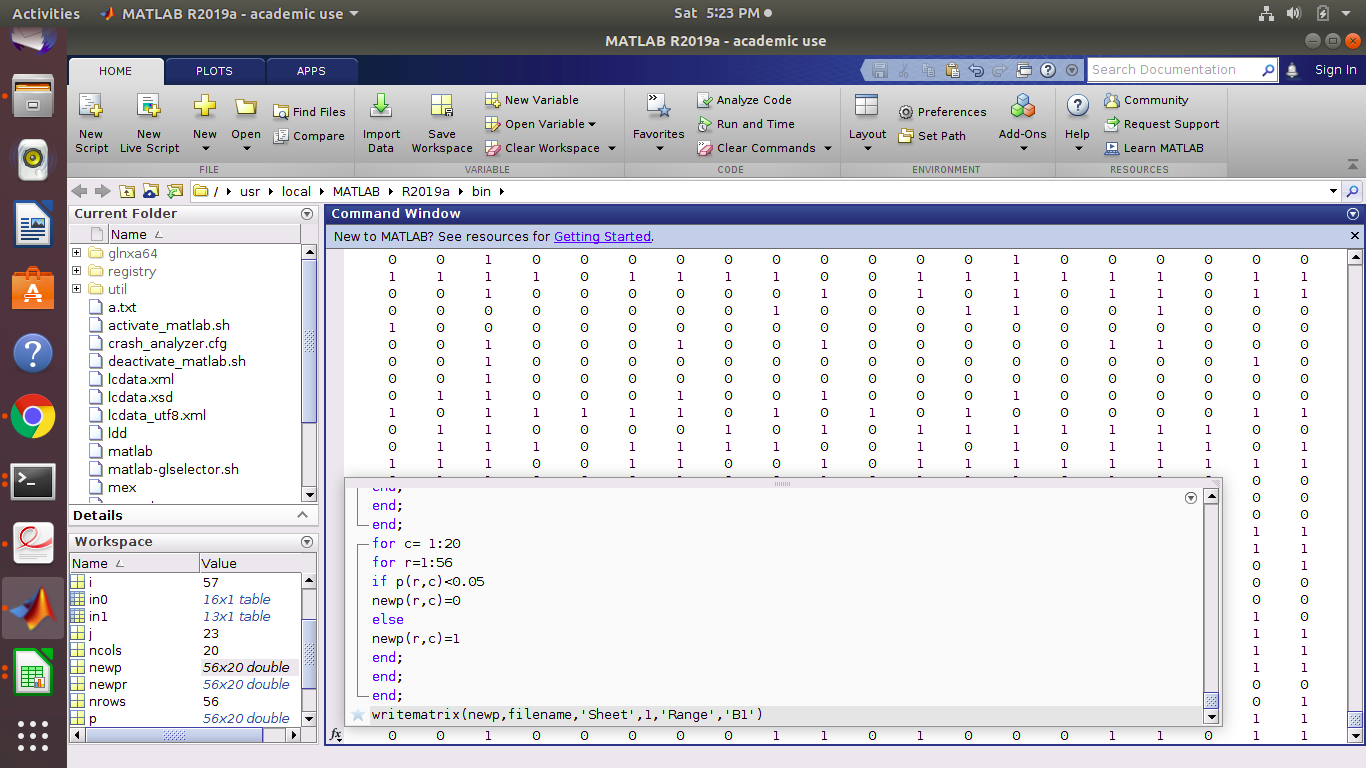
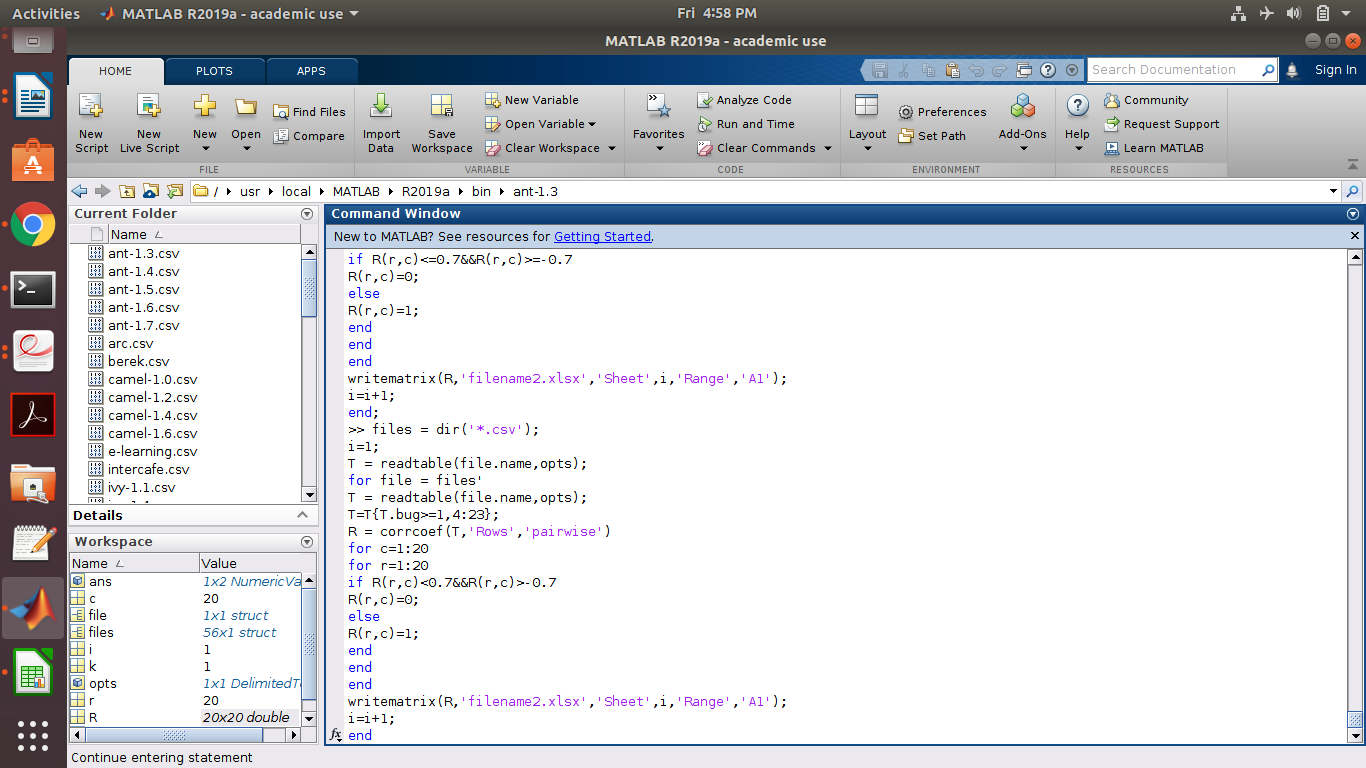


FIG 2: For assigning a value of 0 for p<0.05 and 1 otherwise

**B.Pearson’s Correlation**

FIG 3: For calculating pearson’s correlation and assigning a value of 1 if the source code metric is significant else assign it the value 0.

**Result**

The resulting files are attached with this document and can be seen in figure 4 (for Ranksum test) and figure 5 (for pearson correlation) as well.The WMC, CBO, LCOM, Ca, Ce, LOC, MOA, CAM, AMC, Max\_cc and Avg\_cc source code metrics were found to be significant predictors for fault prediction using ranksum test and WMC, LCOM, CBO, LOC, NPM, MOA, Ce using Pearson’s Correlation. Taking their intersection we come to the result that **WMC, LCOM, CBO, Ce, MOA, LOC** are the best predictors for fault prediction.

**Conclusion**

Using the ranksum test and pearson corrrelation we were able to find out which source code metrics are relevant to us and which are not.



Fig 4:Results**Ranksum Test**



ant-1.3.csv **Fig 5: Pearson Correlation Results**

**Fig 5: Pearson Correlation Results**

ant-1.3.csv

|  |  |
| --- | --- |
| ant 1.4.csv | ant -1.5 csv |
| ant -1.6.csv  **Fig 5: Pearson Correlation Results** | ant-1.7.csv |
| arc.csv | berek.csv |
| camel-1.0.csv  **Fig 5: Pearson Correlation Results** | camel-1.2.csv |
| Camel-1.4.csv | Camel-1.6.csv |
| E-learning.csv  **Fig 5: Pearson Correlation Results** | Intercafe.csv |
| iv-1.1.csv | iv-1.4.csv |
| iv-2.csv  **Fig 5: Pearson Correlation Results** | Jedit-3.2.csv |
| Jedit-4.0.csv | Jedit-4.1..csv |
| Jedit-4.2.csv  **Fig 5: Pearson Correlation Results** | Jedit-4.3.csv |
| kalkulator.csv | Log4j-1.0.csv |
| Log4j-1.1.csv  **Fig 5: Pearson Correlation Results** | Log4j-1.2.csv |
| Lucene-2.0.csv | Lucene-2.2.csv |
| Lucene-2.4.csv  **Fig 5: Pearson Correlation Results** | nieruchomosci.csv |
| pdftrnslator.csv | Prop-1.csv |
| Prop-2.csv  **Fig 5: Pearson Correlation Results** | Prop-3.csv |
| Prop-4.csv | Prop-5.csv |
| Prop-6.csv  **Fig 5: Pearson Correlation Results** | redaktor.csv |
| serapion.csv | skarbonka.csv |
| sklebagd.csv  **Fig 5: Pearson Correlation Results** | synapse-1.0.csv |
| synapse-1.1.csv | synapse-1.2.csv |
| systemdata.csv  **Fig 5: Pearson Correlation Results** | systemdata.csv |
| termoproject.csv | tomcat.csv |
| velocity-1.5.csv  **Fig 5: Pearson Correlation Results** | velocity-1.6.csv |
| workflow.csv | wspomaganiepi.csv |
| xerces-1.2.csv  **Fig 5: Pearson Correlation Results** | xerces-1.3.csv |
| xerces-1.4.csv | xerces-init.csv |
| zuzel.csv  **Fig 5: Pearson Correlation Results** |  |

**References**

1.S.R. Chidamber, C.F. Kemerer, A metrics suite for Object-Oriented design, IEEE Trans. Softw. Eng. 20 (6) (1994) 476–493.

2. R. Martin, Object-oriented design quality metrics an analysis of dependencies, in: Proceedings of Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, 1994.

3. J. Bansiya, C.G. Davis, A hierarchical model for Object-Oriented design quality assessment, ACM Trans. Program. Lang. Syst. 128 (1) (2002) 4–17.

4. B. Henderson-Sellers, Software Metrics, Prentice-Hall, UK, 1996.

5. M. Halstead, Elements of Software Sciencel, Elsevier Science, New York, USA, 1977.

6. M.-H. Tang, M.-H. Kao, M.-H. Chen, An empirical study on object-oriented metrics, in: Software Metrics Symposium, Boca Raton, Florida, USA. Proceedings.Sixth International, IEEE, 1999, pp. 242–249.

7. T.J. McCabe, A complexity measure, IEEE Trans. Softw. Eng. 2 (4) (1976) 308–320.

8. Mendel Suchmacher, Mauro Geller, in Practical Biostatistics, 2012

9. J.-C. Chen, S.-J. Huang, An empirical analysis of the impact of software develop-ment problem factors on software maintainability, J. Syst. Softw. 82 (6) (2009)981–992.

10.Lov K, Sanjay M, Santanu R (2017) An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes. Comput Stand Interfaces. Doi: [10.1016/j.csi.2017.02.003](https://doi.org/10.1016/j.csi.2017.02.003)