A Statistical Comparison of Java and Python Software Metric Properties

Giuseppe Destefanis[1], Marco Ortu[2], Simone Porru[2], Stephen Swift[1], Michele Marchesi[2]
[1]Brunel University London, Uxbridge, United Kingdom
{giuseppe.destefanis,stephen.swift}@brunel.ac.uk
[2]DIEE, University of Cagliari, Italy
{marco.ortu,simone.porru,michele}@diee.unica.it

ABSTRACT

This paper presents a statistical analysis of 20 opens ource object-oriented systems with the purpose of detecting differences in metrics distribution between Java and Python projects. We selected ten Java projects from the Java Qualitas Corpus and ten projects written in Python. For each system, we considered 10 class-level software metrics.
We performed a best fit procedure on the empirical distributions through the log-normal distribution and the double Pareto distribution to identify differences between the two languages. Even though the statistical distributions for projects written in Java and Python may appear the same for lower values of the metric, performing the procedure with the double Pareto distribution for the Number of Local Methods metric reveals that major differences can be noticed along the queue of the distributions. On the contrary, the same analysis performed with the Number of Statements metric reveals that only the initial portion of the double Pareto distribution shows differences between the two languages. In addition, the dispersion parameter associated to the log-normal distribution fit for the total Number Of Methods can be used for distinguishing Java projects from Python projects.

1. INTRODUCTION

With the advent of the object-oriented approach, specific measures have been introduced to evaluate the quality of software systems. The first attempt in this direction was Chidamber and Kemerer's metrics, which have become the most popular Object Oriented metrics suite. The process of defining new measures is still a vibrant research area, but theoretical reasons supporting the adoption of specific metrics is not enough. Software engineers need to have empirical evidence that these metrics are actually related to software quality. In general software quality has many different meanings, and it is associated with practices that lead to software products that are accurate, effective, delivered on time and within budget.

It is still difficult to relate software metrics to the phenomena we want to improve. Additionally, in order to reduce software complexity, new development methodologies and tools are being introduced.

To the best of our knowledge, little has been written about the distribution of traditional metrics in software developed in Python (this programming language has been recently growing in popularity [7, 8]). Conversely, several studies had been made considering Java. In this paper, we present the results about software metrics distributions on 20 open source software system, in order to study possible differences related to the programming language used. We identified a set of Java and Python projects suitable for a comparison of their metrics using distribution functions and we analysed the metrics distributions for the two sets of systems. The remainder of this paper is structured as follows: In the next section, we provide background and related work. Section 3 describes the methodology used for this study. In Section 4, we present the results. Section 5 discusses the threats to validity. Finally, we summarise the study findings in Section 6.

WETSoM'16, May 16 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4177-6/16/05. . . $15.00
http://dx.doi.org/10.1145/2897695.2897697

2. BACKGROUND AND RELATED WORK

Several studies have analysed software metric distributions with regard to study software quality and to define a methodology for guiding the software process development. Many empirical studies have been performed to validate empirically different software metrics for those purposes. The Chidamber and Kemerer (CK) [4] suite tackles these two aspects, showing an acceptable correlation between CK metrics values and software fault-proneness and difficulty of maintenance [2, 3, 18, 20]. Other metrics like micro-patterns have also been proposed to help managers and developers in monitoring software quality [5, 9].

Louridas et al. [10] found that distributions with long, fat tails in software are much more pervasive than previously established, appearing at various levels of abstraction, in diverse systems and languages.

Alshayeb et al. [1] found that Object Oriented metrics are effective in predicting design efforts and source lines of code added, changed, and deleted in the short-cycled agile process and ineffective in predicting the same aspects in the long-cycled framework process.

Potanin et al. [13] examined the graphs formed by object-oriented programs written in a variety of languages, and found that these turn out to be scale-free networks.

Shatnawy et al. [16] validated the effect of power laws on the interpretation of software metrics. The authors have studied five open-source systems to investigate the distribution and their effect on fault prediction models. The results show that power law behaviour has an effect on the interpretation and usage of software metrics and in particular the CK metrics. Many metrics have shown a power law behaviour. Threshold values are derived from the properties of the power law distribution when applied to open-source systems. The properties of a power law distribution can be effective in improving the fault-proneness models by setting reasonable threshold values.

Concas et al. [6] presented an extensive analysis of software metrics for 111 Object Oriented systems written in Java. For each system, authors considered 18 traditional metrics such as LOC and CK metrics, as well as metrics derived from complex network theory and social network analysis; they also considered two metrics at system level, namely the total number of classes and interfaces and the fractal dimension. They discuss the distribution of these metrics and their correlation both at class and system level. They found that most metrics followed a leptokurtotic distribution. Only a couple of metrics have patent normal behaviour while three others are very irregular and even bimodal.

Different software metrics may follow different statistical distributions. In particular, there are metrics whose values may differ largely from those of other metrics, like the DIT (depth of inheritance tree), which typically assumes values lower than ten. In our study, we did not consider such metrics because the related statistics can hardly be useful to distinguish one language from the other. We focused on metrics whose distributions have wider ranges, since this choice increases chances of finding differences between the two languages. Typically these software metrics follow an approximate power law in the tail of the distribution and can be fitted by many statistical distributions.

Among all the candidate distribution functions, we chose only two of them. We searched for distributions capable of providing not only the best fit for all the projects and the analysed metrics, but which could also yield significant results suitable for revealing differences among the metrics of choice. The statistical distributions we employed are the log-normal and the double Pareto distribution.

The double Pareto distribution is an extension of the standard power-law, in which two different power-law regimes exist, provided by the two parameters α and β in eq. 1. There are different forms for the double Pareto distribution in literature [11, 14, 15, 17]. We use the form described in [17] for the Complementary Cumulative Distribution Function (CCDF), because it provides a smoother transition across the two different power-law regimes:

$$P(x) = 1 - \left[\frac{1 + (m/t)^{-\alpha}}{1 + (x/t)^{-\alpha}}\right]^{\beta/\alpha} \tag{1}$$

The double Pareto distribution is able to fit a power-law tail in the distribution, with the advantage of being more flexible than a single power-law in fitting also the head of the distribution, where it is very similar to a log-normal.

3. METHODOLOGY

In the first step we have selected active projects from the Java Qualitas Corpus [19], subsequently trying to identify similar Python projects, considering system dimension in terms of number of classes, number of recent commits and application domain [12]. We decided to select open source projects and, to ensure uniformity among the systems, we have considered only projects with similar features.

The number of classes for the projects considered in our corpus ranges from the minimum value of 3357 (FreeCol, Python) to the maximum value of 31604 (Vuze, Java). Considering the selection criteria, during the systems selection process we have considered two features: popularity and high activity. The former concept is related to the total number of users of the specific software, the latter to the number of commits. We have considered the statistics provided by openhub.net. On this portal it is possible to compare a huge number of open source projects in terms of popularity and in terms number of commits related to a specific project. We have considered systems belonging to the same area; to do that we have analysed the metadata provided by the Qualitas Corpus, openhub.net, and other detailed information available on the official websites of each project present in our corpus.

The systems we investigated are the following:

Table 1: Python and Java Systems

Python	Java
Biopython	Apache Ant
Calibre	Apache Commons Collections
Matplotlib	Apache JMeter
Open Edx - platform	Castor
OpenERP	Cayenne
Py-Pandas	Freecol
Sage	Jena
SCons	JFreeChart
Tribler	Vuze
Unknown Horizons	Weka

We used Understand 3.1 (build 766)[1] to calculate a set of nine metrics (Number of base classes, Number Of Declared Instance Methods, Number of Declared Instance Variables, Number Of Local Methods, Total number of Methods, Number of Lines of code, Number of Lines of comments, Number of statements, Depth of inheritance tree) from each of the twenty systems, computed the empirical cumulative distribution function, and employed the Log-normal and the double Pareto distributions to identify differences between the metrics. The statistical analysis on the results were performed using R[2] and Matlab[3].

After computing the previously listed metrics, we only focused on three of them, namely Number Of Local Methods (NOLM), Number of Methods (NOM) and Number Of Statements (NOS), as they proved to yield the most meaningful results. We have proceeded the following way. We have performed best fitting procedure on the metrics empirical distributions by employing the two chosen statistical distributions functions, the log-normal and the double

[1]Understand. Scitools.com: https://scitools.com
[2]The R Project for Statistical Computing - www.r-project.com
[3]www.mathworks.com

Pareto distributions, for both the ten Java projects and for the ten Python projects. We have systematically verified the goodness of fit for all of them, provided by R^2, the goodness of fit coefficient, which ranges from 0 to 1. When R^2 is close to 1, the fit is very good. When R^2 is larger than 0.9, the fit is still considered fairly statistically good, while smaller values provide a worse fit. We have obtained the values of the best fitting parameters across the twenty analysed projects, which statistically represent the metrics values for each system. These parameters are μ and σ for the log-normal distribution, and the α and β parameters in the case of the double Pareto distribution. We then statistically compared the two sets of measures, those related to the Java projects and those related to the Python projects. We did not only measured mean values and standard deviations for all the parameters using the ten measures for the two sets of projects, but also performed statistical tests for measuring whether the two sets belong to two different distributions. These are the rank sum test, and the Kruskal-Wallis test, both non parametric, in order to avoid any arbitrary assumption on the distribution function of the parameters across the different projects.

4. RESULTS

Most of the empirical distributions resulting from the analysis, given a specific metric, are at a first glance, quite similar to each other. Nevertheless, some of them show that differences between Java and Python projects do exist. In particular, four metrics appeared to be suitable candidates:

- the number of methods that can be called on an instance of the class in which they are declared (Number Of Declared Instance Methods, NODIM);
- the number of methods that can be called on a class and on an instance of the class in which they are declared (Number Of Local Methods, NOLM);
- the total number of methods (NOM), inherited methods included;
- the number of statements (NOS);

The first two metrics are capable of distinguishing between instance methods and static methods, since NODIM accounts only for the methods that can be called on an instance of the class, whereas NOLM accounts both for those methods and those that can be called directly in the class. Python projects present the same values for NODIM and NOLM in all classes, since instance methods, static methods and class methods can all be called on an instance of a Python class. Thus, NODIM and NOLM analysis yield the same results, as far as Python projects are concerned. We then considered only the last three metrics, i.e. NOLM, NOM and NOS.

We first analysed the results obtained with NOLM metric. The NOLM Empirical Cumulative Distribution Function (ECDF) shows that the Log-normal distribution fits for Python and Java systems, as the coefficient of determination is close to one for all the projects, with an average of 0.983 and of 0.985 for the ten Java projects and the ten Python projects respectively. The fitting procedure provides different averages for the location parameter, thus showing a difference between projects written in one language with respect to the other. In particular, if we look at the mean value of the location parameter for each set of projects, Python projects show a lower value. Table 2 shows the comparison between average values for the Log-normal distribution parameters we extracted.

Table 2: NOLM ECDF Averaged Log-normal fit parameters with standard deviation

	μ	σ	R^2
Java	1.369 ± 0.297	1.048 ± 0.110	0.983 ± 0.015
Python	1.271 ± 0.231	0.990 ± 0.093	0.985 ± 0.006

Even if the average values for parameter μ are apparently dissimilar, their large standard deviation suggests a possible overlap among the values of μ for projects belonging to the two languages. On the contrary, the parameter σ displays two fairly close average values, but their standard deviations are quite small and the overlap among the two sets appears reduced.

We performed the rank sum test and the Kruskal-Wallis statistical test on the twenty previously computed values of μ and of σ, ten for each programming language, in order to establish if the two sets show statistical differences between their metrics.

For the log-normal distribution both tests provide a negative answer. Namely, the parameters μ and σ, obtained from the best fitting with a log-normal distribution for the metric NOLM do not allow to distinguish between Java and Python projects, as table 3 shows. This result confirms that the position parameters present a good overlap between the two languages, and that the dispersion parameter sigma, even if it presents a smaller overlap, does not allow to statistically distinguishing between Java and Python using the metric NOLM.

Table 3: Rank sum and Kruskal-Wallis tests results for NOLM Log-normal fits

	μ	σ
Rank sum (p-value)	0.3075	0.0890
Kruskal-Wallis (p-value)	0.2899	0.0821

The situation changes in the case of the double Pareto distribution (see fig.1 and 2).

Table 4 reports the average values and the standard deviations for α and β best fitting parameters obtained for the metric NOLM. Also in this case the coefficient of determination R^2 is very close to one, meaning that the fitting with a double Pareto distribution is good. In this case the advantage is that the parameter β averages obtained from Java projects are different enough from Python ones, as reported in table 4

Table 4: NOLM ECDF Averaged double Pareto fit parameters with standard deviation

	α	β	R^2
Java	1.8993 ± 0.3031	1.2583 ± 0.1797	0.9868 ± 0.0141
Python	1.8917 ± 0.2151	1.4383 ± 0.0845	0.9894 ± 0.0052

This consideration is confirmed by the rank sum and Kruskal-Wallis tests (table 5), which reveal that differences in the

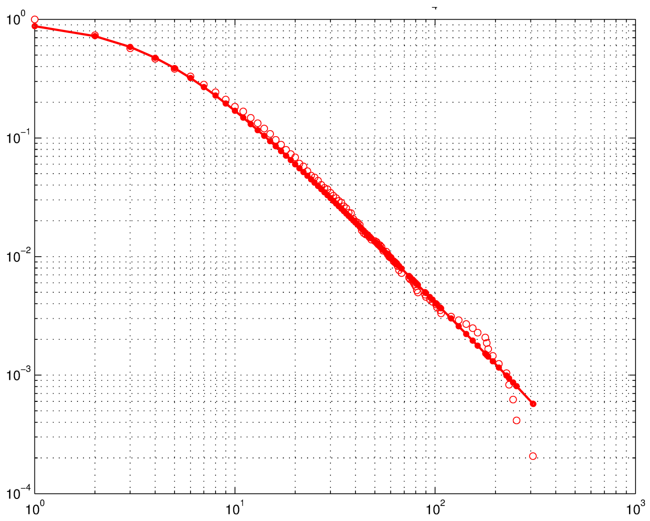

Figure 1: NOLM ECDF with double Pareto fit for Jena 2.11.1 (Java)

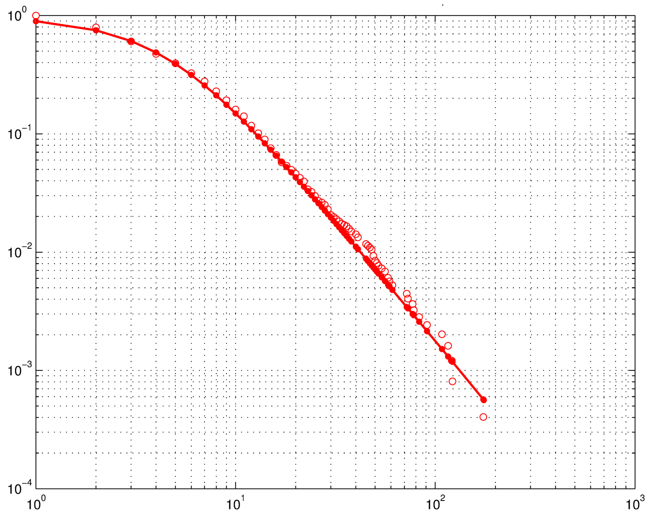

Figure 2: NOLM ECDF with double Pareto fit for Calibre 1.18.0 (Python)

parameter β are statistically meaningful for distinguishing Java projects from Python ones, even if the parameter α does not show differences between the two languages.

Parameter α provides the power-law exponent in the first power-law regime, whereas parameter β provides the power-law exponent for the second regime. Therefore, the obtained results show that, even though the statistical distributions may appear the same in the bulk of the analysed data, that is, for classes with NOLM metric below the first regime threshold and ranging from 5 to 6 both for Java and for Python, major differences are evident along the queue of the distributions, namely for higher values of the NOLM metric. Thus, the use of local methods in Java and Python classes is different for classes having a large number of local methods. A difference between metric distributions for Python and Java projects has been found, as far as NOLM metric is concerned.

Table 6 reports the same results for the best fitting parameters of the log-normal distribution with respect to the metric NOM.

Table 5: Rank sum and Kruskal-Wallis tests results for NOLM double Pareto fits

	α	β
Rank sum (p-value)	0.7337	0.0257
Kruskal-Wallis (p-value)	0.7055	0.0233

Table 6: NOM ECDF Averaged Log-normal fit parameters with standard deviation

	μ	σ	R^2
Java	2.122 ± 0.430	1.451 ± 0.180	0.948 ± 0.044
Python	2.42 ± 0.387	1.322 ± 0.125	0.945 ± 0.061

The log-normal distribution provides a worse fitting for the NOM metric ECDF than the previous case, being the coefficient of determination below 0.95 on average for projects in both languages. Nevertheless, the analysis suggests that the related dispersion parameter can be used for distinguishing Java projects from Python projects, since the relatively small standard deviations for σ for both Java and Python projects do not show a consistent overlap between the two averages. In figure 3 and 4 we plot the NOM ECDF and related log-normal fit for Apache Commons Collection and Pandas, a Java and a Python project respectively.

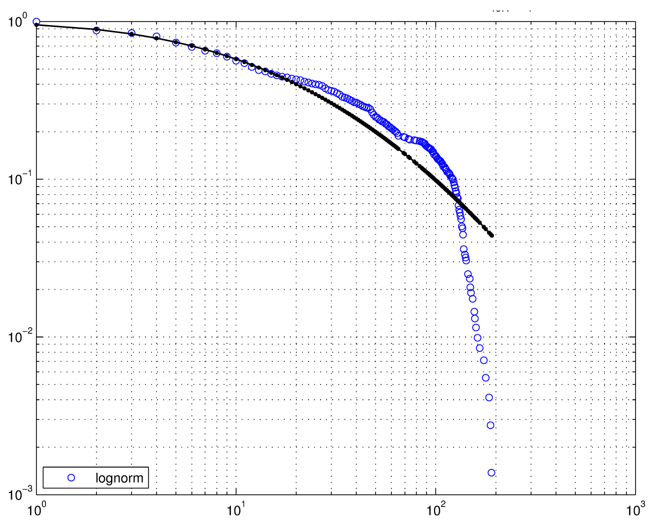

Figure 3: NOM ECDF with log-normal fit for Apache Commons Collections 4-0-RC5 (Java)

We have verified this result performing rank sum and Kruskal-Wallis tests for the distributions of μ and σ relative to the ten Java projects and the ten Python projects, as we previously did with the NOLM metric. Test results are displayed in table 7 and show that the dispersion parameter σ can be used for distinguishing between Java and Python projects.

We have applied the best fitting procedure for the NOM ECDF also using the double Pareto statistical distribution, obtaining results similar to those obtained through the log-

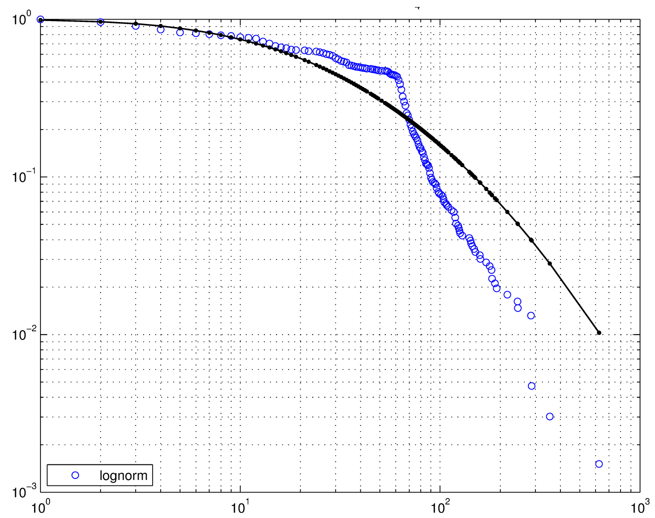

Figure 4: NOM ECDF with log-normal fit for Pandas 0.13.1 (Python)

Table 7: Rank sum and Kruskal-Wallis tests results for NOM Log-normal fits

	μ	σ
Rank sum (p-value)	0.1405	0.0452
Kruskal-Wallis (p-value)	0.1306	0.0413

normal, as far as the coefficient of determination is concerned; nevertheless, they are useless with regard to the capability of distinguishing between Java and Python projects. Table 8 reports the results for the best fitting parameters averages and the coefficient of determination averages for the double Pareto case for the NOM metric, as well as the relative standard deviations.

Table 8: NOM ECDF Averaged double Pareto fit parameters with standard deviation

	α	β	R^2
Java	1.0728 $\pm$ 0.2107	2.6849 $\pm$ 4.6332	0.9525 $\pm$ 0.0439
Python	1.1227 $\pm$ 0.1844	1.8015 $\pm$ 0.7450	0.9441 $\pm$ 0.0630

In this case it is clear that the high variability of parameter β for both languages, related to the shape of the NOM ECDF, renders it useless for distinguishing between the two languages, even if the two averages are quite different. Rank sum and Kruskal-Wallis tests confirm this result, according to table 9.

On the other hand, examining the possible outliers, the project "JFreeChart" in Java results in a β value of 15.8542, which is clearly out of scale, considering that the maximum of the remaining values is around 1.7. Repeating the analysis after removing such outlier reduces the β variance in Java, producing more interesting results. In fact, in such case the rank sum test provides a p-value of 0.0133 for the comparison between the two sets of β values, and the Kruskal-Wallis test applied using nine Java and Python projects provides a p-value of 0.0243, indicating that parameter β holds the capability of distinguishing between the two languages.

Table 9: Rank sum and Kruskal-Wallis tests results for NOM double Pareto fits

	α	β
Rank sum (p-value)	0.3847	0.0640
Kruskal-Wallis (p-value)	0.3643	0.0588

Finally, NOS ECDF show more significant differences between Java and Python languages than NOM ECDF (see figure 5.

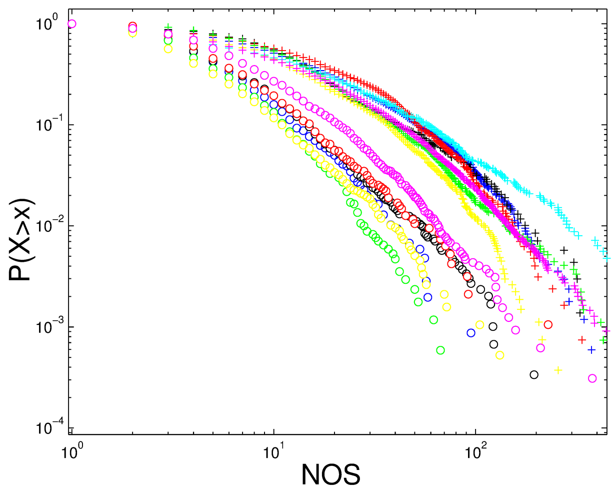

Figure 5: NOS ECDF for eight Python projects (O) and eight Java projects (+)

The average value for the coefficient of determination is very high for both Python and Java, and for log-normal as well as double Pareto best fits. The log-normal location parameters are dissimilar: the average value for Java systems is around 2.23, whereas for Python is around 1.44. Mean and standard deviation for Java and Python projects are presented in table 10.

Table 10: NOS ECDF Averaged Log-normal fit parameters with standard deviation

	μ	σ	R^2
Java	2.225 $\pm$ 0.360	1.278 $\pm$ 0.186	0.991 $\pm$ 0.007
Python	1.444 $\pm$ 0.215	0.940 $\pm$ 0.114	0.989 $\pm$ 0.004

The rank sum and Kruskal-Wallis tests provide meaningful results for both μ and σ, according to table 11, which demonstrates Java projects are distinguishable from Python projects through the Log-normal distribution.

In figure 5 the ECDF for eight Python projects and eight Java projects are illustrated.

Conversely, the results obtained with the double Pareto distribution show that only parameter α allows to distinguish between the two languages, while the parameter β is useless (see table 12).

This means that, unlike the NOLM metric, solely the initial portion of the double Pareto distribution makes it possible to differentiate between the two languages. Namely, the statistical distributions appear different in the bulk of the

Table 11: Rank sum and Kruskal-Wallis tests results for NOS Log-normal fits

	μ	σ
Rank sum (p-value)	< 0.00044	< 0.00044
Kruskal-Wallis (p-value)	< 0.0004	< 0.0004

Table 12: NOS ECDF Averaged double Pareto fit parameters with standard deviation

	α	β	R^2
Java	1.2857 ± 0.1634	1.6221 ± 0.6256	0.9898 ± 0.0063
Python	1.9978 ± 0.3180	1.7224 ± 0.3161	0.9920 ± 0.0032

analysed data, that is, for classes having NOS values below the first regime threshold, which is around 8 for Java and Python, while they are similar along the queue of the distributions, namely for larger value of NOS metric. Thus the use of statements in Java and Python classes appears to be different for classes having a small number of local methods. Table 13 shows the results for rank sum and Kruskal-Wallis tests for the double Pareto parameters for metric NOS.

Table 13: Rank sum and Kruskal-Wallis tests results for NOS double Pareto fits

	α	β
Rank sum (p-value)	0.0010	0.1859
Kruskal-Wallis (p-value)	8.8074e-04	0.1736

These results show that the metric Number of Statements appears the most different in Java and Python projects, since they can be easily distinguished using both best fitting distributions, and in both cases the fit is fairly good.

Considering the obtained results, we succeeded in identifying 10 Java projects and 10 Python projects having comparable dimensions, belonging to similar application domains, and presenting similar statistical distributions for the corresponding metrics. We were also able to identify statistical distributions providing fairly good fits for the empirical distributions of their metrics, for all Java and Python projects. Finally, the best fitting parameters have been successfully used for a comparison of the metric distributions.
We found that systems developed in Java and Python do present different metrics distributions.
The double Pareto distribution fit for NOM metric reveals that, even though the statistical distributions for projects written in Java and Python may appear the same for lower values of the metric, major differences can be noticed along the queue of the distributions. The same analysis, performed with NOS metric, reveals that only the initial portion of the distribution shows differences between the two languages. Finally, the dispersion parameter associated to the log-normal distribution fit for NOM metric could also be used for distinguishing Java projects from Python ones.

The number of classes across the ten Java projects and the ten Python projects varies consistently from one system to the other, but the average number of classes for Java projects is twice the average number of classes for Python projects. Nevertheless, the average number of lines of code is roughly the same in both cases. This indicates that on average the number of lines of code in Python classes is roughly twice the corresponding number in Java classes. Thus Python projects use fewer classes than Java projects, but the same number of lines of code, on average. On the other hand, our data for the NOS metric show that the average number of statements declarations per class is larger in Java, despite the classes contain less lines of code on average. Since each statement normally requires one line of code, these results seem contradictory, and indicate that there must be a specific reason which drives Java software developers to write code richer in statement declarations than Python code.

5. THREATS TO VALIDITY

Some projects were selected even if they are at a stage of their life cycle in which it cannot inherently be possible to notice an appreciable number of recent commits. This happens because they are less prone to show major bug issues, as they are mature systems. So, they can be considered as fully representative for the language used in their development.

In addition to this, it is worth to mention that not every Python system has a correspondent Java system, and vice versa. This is true as far as software scope is concerned. It was possible to identify only a limited subset of projects whose scope could be considered similar to that of each other. So, we are not only considering software pairs that serve almost the same purposes, but also other projects that do not show this relationship.

One of the projects, SCons, might not be qualified as Python project, because of the lower percentage of Python code with respect to that of XSLT. This could be considered as a minor issue, because there is a most relevant difference between Python and XSLT, more relevant than that between Java and Python. Although, it must be added that analyzing the code implies specifically looking for Python files for Python projects and Java files for Java ones.

6. CONCLUSION

We presented a statistical analysis of 20 open-source object-oriented systems (10 written in Java, 10 written in Python), with the purpose of detecting differences between Java and Python projects through the study of the related metrics distributions.

We identified two statistical distribution functions, the log-normal and the double Pareto distributions, capable of providing good fits for three metrics for all Java and Python projects in the corpus; this made it possible to compare the distributions of the corresponding metrics for the two programming languages.

Our analysis showed that the best fitting parameters of these distributions allow for distinguishing between the two programming languages when using NOLM, NOM, and NOS metrics.

In particular, the metric NOLM revealed differences between Java and Python, detected through the parameter β of the double Pareto distribution fitting, meaning that classes with a higher number of local methods are differently used by developers in Java and in Python. The metric NOM showed differences in the dispersion parameter of the log-normal distribution, and also in the parameter β of the double Pareto distribution, meaning that the variability of the number of methods used in Python classes is lower than in Java classes, and also that classes with more methods (not only considering local methods), are differently used in the two languages.

Finally, the metric NOS revealed that statements are differently used in the two languages, and also that the average number of statements declarations in Java is much larger, even if Java classes feature fewer lines of code on average.

In conclusion, our study showed that a metrics analysis can be successfully leveraged to identify differences in programming practices related to the use of methods and statements in Java and Python, thus revealing meaningful differences between the two programming languages.

7. ACKNOWLEDGEMENT

The research presented in this paper was partly funded by the Engineering and Physical Sciences Research Council (EPSRC) of the UK under grant ref: EP/M024083/1.

8. REFERENCES

[1] M. Alshayeb and W. Li. An empirical validation of object-oriented metrics in two different iterative software processes. *Software Engineering, IEEE Transactions on*, 29(11):1043–1049, 2003.

[2] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22(10):751–761, 1996.

[3] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. *Software Engineering, IEEE Transactions on*, 24(8):629–639, 1998.

[4] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[5] G. Concas, G. Destefanis, M. Marchesi, M. Ortu, and R. Tonelli. Micro patterns in agile software. In *Agile Processes in Software Engineering and Extreme Programming: 14th International Conference, XP 2013, Vienna, Austria, June 3-7, 2013, Proceedings*, volume 149, page 210. Springer, 2013.

[6] G. Concas, M. Marchesi, A. Murgia, S. Pinna, and R. Tonelli. Assessing traditional and new metrics for object-oriented systems. In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, pages 24–31. ACM, 2010.

[7] G. Destefanis. Which programming language should a company use? a twitter-based analysis. 2014.

[8] G. Destefanis, S. Counsell, G. Concas, and R. Tonelli. Software metrics in agile software: An empirical study. In *Agile Processes in Software Engineering and Extreme Programming*, pages 157–170. Springer, 2014.

[9] G. Destefanis, R. Tonelli, E. Tempero, G. Concas, and M. Marchesi. Micro pattern fault-proneness. In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, pages 302–306. IEEE, 2012.

[10] P. Louridas, D. Spinellis, and V. Vlachos. Power laws in software. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 18(1):2, 2008.

[11] M. Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, 1(3):305–333, 2004.

[12] M. Orrú, E. Tempero, M. Marchesi, R. Tonelli, and G. Destefanis. A curated benchmark collection of python systems for empirical studies on software engineering. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*, page 2. ACM, 2015.

[13] A. Potanin, J. Noble, M. Frean, and R. Biddle. Scale-free geometry in oo programs. *Communications of the ACM*, 48(5):99–103, 2005.

[14] W. J. Reed and B. D. Hughes. From gene families and genera to incomes and internet file sizes: Why power laws are so common in nature. *Physical Review E*, 66(6):067103, 2002.

[15] W. J. Reed and M. Jorgensen. The double pareto-lognormal distributionâĂŤa new parametric model for size distributions. *Communications in Statistics-Theory and Methods*, 33(8):1733–1753, 2004.

[16] R. Shatnawi and Q. Althebyan. An empirical study of the effect of power law distribution on the interpretation of oo metrics. *ISRN Software Engineering*, 2013, 2013.

[17] C. P. Stark and N. Hovius. The characterization of landslide size distributions. *Geophysical Research Letters*, 28(6):1091–1094, 2001.

[18] R. Subramanyam and M. S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *Software Engineering, IEEE Transactions on*, 29(4):297–310, 2003.

[19] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. The qualitas corpus: A curated collection of java code for empirical studies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 336–345. IEEE, 2010.

[20] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*, pages 531–540. ACM, 2008.