Analysis of advanced aggregation techniques for software metrics

Vasilescu, B.N.

Award date:
2011

Analysis of Advanced Aggregation Techniques for Software Metrics

Bogdan Vasilescu

July 2011

Eindhoven University of Technology
Department of Mathematics and Computer Science

Analysis of Advanced Aggregation Techniques for Software Metrics

Bogdan Vasilescu

in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Supervisor: dr. Alexander Serebrenik

Examination Committee:
prof. dr. Mark G. J. van den Brand
dr. Serguei Roubtsov
dr. Alexander Serebrenik

Eindhoven, July 2011

Acknowledgements

This thesis is the result of my graduation project, which concludes the master program Computer Science and Engineering at the Eindhoven University of Technology. The project was carried out within the Software Engineering and Technology group of the department of Mathematics and Computer Science.

First and foremost I owe my deepest gratitude to my supervisor, Dr. Alexander Serebrenik, who has continuously supported and encouraged me throughout this period. His comments, suggestions, and advice were invaluable to the completion of this work. One simply could not wish for a better or friendlier supervisor.

Moreover, I would like to show my gratitude to the members of the Algorithms group of the department of Mathematics and Computer Science, for endowing me with an algorithmic mindset and for teaching me to be critical and conscientious about my work.

Furthermore, I would like to thank the members of my examination committee, prof. dr. Mark van den Brand and dr. Serguei Roubtsov, for their feedback after my interim presentations and suggestions for improvement of this work. Likewise, I am grateful to Vincent Kusters and Martijn Klabbers for their interest and comments on how to improve the readability of this thesis.

Additionally, I am indebted to the staff of the Laboratory for Quality Software for accommodating me during this period, and for creating such a pleasant and propitious work environment.

Finally, pursuing this master program and writing this thesis would not have been possible without the financial support of Océ-Nederland B.V.

Bogdan Vasilescu

Eindhoven,
July 2011

Abstract

A popular approach to assessing software maintainability and predicting its evolution involves collecting and analyzing code metrics. However, metrics are usually defined on a micro-level (e.g., method, class), and should therefore be aggregated in order to provide insights in the evolution at the macro-level (system). In addition to traditional aggregation techniques such as the *mean*, *median*, or *sum*, econometric aggregation techniques on the one hand , such as the Gini, Theil, MLD, Kolm, Atkinson, and Hoover inequality indices, and threshold-based aggregation techniques on the other hand, such as the approaches in the quality models set forth by the SIG company or Squale consortium, have been proposed and applied to software metrics.

In this thesis we study several main econometric and threshold-based aggregation techniques for code metrics, with the goal of distilling requirements for future aggregation techniques for software metrics. We perform our study along two directions. In the first part we assume a theoretical standpoint and study properties of these techniques relevant to aggregation of code metrics. Additionally, we show that root-cause analyses can be performed efficiently using the Theil index, and that the aggregation technique in Squale shares common properties with inequality indices, and has a formal relation to the Kolm index.

In the second part we present the results of a series of empirical studies of all three categories of aggregation techniques considered (i.e., traditional, econometric, and threshold-based) using different metrics and aggregation levels. For example, we observe consistently high and statistically significant correlation between the Gini, Theil, MLD, Atkinson, and Hoover inequality indices for all metrics considered, i.e., aggregation values obtained using these techniques convey the same information regardless of the metric. However, even though it might seem that all these indices are equally appropriate, this is not true since different indices have different application domains, emphasize different dimensions of inequality and possess different decomposability properties.

Finally, based on our theoretical and empirical analyses, in the third part we propose requirements for future aggregation techniques for software metrics.

Contents

Chapter 1

Introduction

Measurement and metrics have long been regarded by software practitioners as useful and necessary [25, 42, 75, 89]. Recently, metrics have also been recognized as part of the software development fabric, essential to understanding whether the quality of the software we are building corresponds to our expectations [88]. Thus, many different metrics have been proposed, as well as a plethora of tools to compute them and perform quality assessments.

Considering the different stakeholders participating in software projects (e.g., developers, managers, users, etc.), quality needs to be evaluated at different levels of detail. For example, *developers* use metrics to assess whether the implementation meets the design quality guidelines and to identify potential areas of improvement [96]. In contrast, *project managers* use metrics to evaluate the project status and assess whether, e.g., the budget will be exceeded, while *maintainers* use metrics to assess which modules should be refactored and what should be upgraded and improved [39].

However, metrics are usually defined at micro level (method, class), while analyses of quality aspects such as maintainability require insights at macro (system) level. Moreover, due to privacy reasons, it might be undesirable to disclose metrics data pertaining to a single developer as opposed to that pertaining to the entire project [102]. Metrics should therefore be *aggregated*.

We distinguish between two types of aggregation of software metrics. On the one hand, aggregation can be performed on values obtained by applying different metrics to the same software artifacts. For example, the *Maintainability Index* (MI) [83] aggregates the *Halstead Effort*, *McCabe's cyclomatic complexity*, the *number of lines of code*, and the *number of comment lines*. Similarly, the *Modularization Quality* (MQ) [69] aggregates intra-connectivity (*cohesion*) and inter-connectivity (*coupling*).

On the other hand, aggregation can be performed on values obtained by applying the same metric to different software artifacts, which is also the focus of this thesis. For example, *Weighted Methods per Class* (WMC) [22] is the sum of McCabe's cyclomatic complexities of methods defined in a class,

while *average number of lines of code* for a class is the arithmetic mean of the number of lines of code of all the nested methods. Note that the two types of aggregation are not mutually exclusive, and often the former encompasses the latter. For example, when computing the *Maintainability Index*, one needs to compute the *average number of lines of code per module* and the *average cyclomatic complexity* first.

When focusing on aggregation of values obtained by applying the same metric to different software artifacts, popular aggregation techniques include additive measures (*sum*), central tendency measures (*mean*, *median*), statistical dispersion measures (*standard deviation*, *variance*), or distribution shape measures (*skewness*, *kurtosis*) [12,63,64,87]. Their main advantage is universality (metrics-independence), i.e., the measures should be calculated in the same way regardless of the metrics being considered.

However, as the distribution of many interesting software metrics is skewed [112], the interpretation of central tendency measures (*mean*, *median*) becomes unreliable: the *mean* can be misleading for highly skewed distributions due to influence of outliers, while the *median*, although less sensitive to outliers, can yield different results if a small change occurs in the data set, e.g., one value is removed [114, 115]. Similarly, since both the *standard deviation* and the *variance* are based on the *mean*, they also become unreliable for highly skewed distributions, where they do not convey information about the asymmetry. On the other hand, albeit easily computable, the *sum* loses the absolute value information and it is unbounded, making relative comparisons difficult. *Skewness* and *kurtosis* suffer from the same drawback, i.e., they are unbounded, and therefore cause difficulties when comparing systems with different population sizes [113].

Alternatively, *distribution fitting* [23,98,112] consists of selecting a known family of distributions (e.g., log-normal or exponential) and fitting its parameters to approximate the metric values observed. The fitted parameters can then be seen as aggregating these values. However, the fitting process should be repeated with each new metric considered. Moreover, it is still a matter of controversy whether, e.g., software size is distributed log-normally [23] or double Pareto [52]. We do not consider distribution fitting.

Recently, there is an emerging trend in using more advanced aggregation techniques borrowed from econometrics, where they are used to study inequality of income or welfare distributions [29–31]. The motivation for applying such techniques to software metrics is twofold.

First, as numerous countries have few rich and many poor, numerous software systems have few very big (complex) components, and many small (simple) ones [14,46,113]. Thus, it is common both for software metrics, and for econometric variables to have strongly-skewed distributions (Figure 1.1).

Second, the shape of these distributions, which appear visually to follow a power law, renders the use of traditional aggregation techniques such as the *sample mean* and *variance* questionable at best. Indeed, it was reported

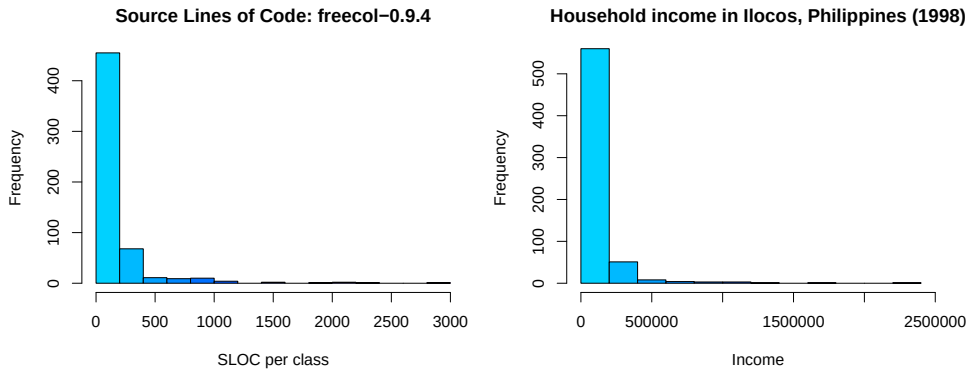

Figure 1.1: Software metrics (SLOC) and econometric variables (household income in the Ilocos region, the Philippines [121]) have distributions with similar shapes.

that many important relationships between software artifacts follow a power-law distribution [23, 118], and it is known that a power-law distribution may not have a finite *mean* and *variance* [14]. These realizations led to the application of econometric techniques to aggregation of software metrics [46, 99, 113–115], and to our current interest in these aggregation techniques.

Nonetheless, although reliable for highly-skewed distributions, income inequality indices have not been designed for software metrics, and there are situations in which they cannot be applied to such data. For example, the presence of a logarithm in the definition of the *Theil index* hinders its application to negative values (e.g., the *Maintainability Index* ranges from 171 to an unbounded negative number), for which the logarithm is undefined.

Concomitant to studies of aggregation of software metrics by means of econometric inequality indices, new techniques for aggregation of the same metric applied to different artifacts have been proposed in the *SIG* [5] and *Squale* [76] software quality models. For example, in contrast to inequality indices, aggregation in Squale starts with the translation of each metric value to an *individual mark* in the range $[0, 3]$, such that clearly desirable values get the highest mark (3). The translation function is chosen such that when a certain threshold is exceeded, the individual mark decreases following an exponential curve. Individual marks are then aggregated to a *global*, system-level mark. Nonetheless, for the Squale approach to be applicable, metrics values should be first transformed to individual marks in the range $[0, 3]$, and threshold values for different metrics should be known. Such thresholds have been published by the Squale project [10], or can be derived empirically from measurement data using the approach proposed by Alves et al. [6]. Nevertheless, a more extensive threshold validation is desirable. Moreover, a theoretical and empirical comparison between the Squale approach and the econometric inequality indices was, so far, missing.

In this thesis we study advanced aggregation techniques for software metrics, such as the inequality indices and the approaches proposed in the SIG and Squale quality models, with the goal of distilling requirements for future aggregation techniques for software metrics.

The thesis is organized in three parts. In the first part, comprising Chapters 2-4, we perform a theoretical analysis of three categories of existing aggregation techniques, and discuss their applicability to software metrics. In the second part of the thesis, consisting of Chapters 5-8, we perform an empirical analysis of the aggregation techniques in part one. Finally, in the third part, comprising Chapter 9, based on insights derived from both the theoretical and the empirical analyses in the first two parts, we identify requirements for aggregation of software metrics.

Specifically, Chapter 2 discusses traditional aggregation techniques such as *mean*, *sum*, or *median*. Chapter 3 introduces econometric inequality indices such as the *Gini* or the *Theil* indices, and discusses their applicability to software metrics. In a recent study [5], the authors claim that inequality indices do not provide means of identifying the underlying measurements (single values) which explain the computed inequality. In contrast, our main contribution in Chapter 3 shows that root-cause analyses are indeed possible using the Theil index, and can be performed efficiently (Lemma 3.2.1).

Chapter 4 discusses threshold-based approaches such as the ones proposed in the *SIG* and *Squale* quality models. Our main contributions are the study of mathematical properties of the aggregation technique in Squale, and the comparison of these properties to mathematical properties of the inequality indices.

Next we proceed with the empirical evaluation of the aggregation techniques. Specifically, in Chapter 5 our main contributions are two pilot studies, conducted in order to assess the feasibility of performing similar large-scale studies, and to distill requirements for the tooling to facilitate them. Appendix A describes the proposed tooling satisfying these requirements.

Later, in Chapter 6 our main contribution is an extensive empirical comparative study of the traditional aggregation techniques and the inequality indices applied to aggregating different code metrics from class to package level. We determine which and to what extent the aggregation techniques agree, i.e., rank distributions of metric values similarly, and we study the evolution of the statistical correlation between the various techniques in time, as the systems evolve. In Chapter 7 we change the aggregation level to method–class and perform a similar correlation study, extended to enable the comparison with the threshold-based aggregation techniques. In Chapter 8 our main contribution is a direct empirical comparison between the inequality indices and the aggregation technique proposed in Squale.

Finally, we conclude the previous theoretical and empirical analyses with the identification of requirements for aggregation techniques for code metrics in Chapter 9.

1.1 Publications arising from this thesis

The work described in this thesis has been published as described in the following list:

1. Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. Comparative study of software metrics aggregation techniques. In Stephane Ducasse, Laurence Duchien, and Lionel Seinturier, editors, *9th Belgian-Netherlands Software Evolution Seminar* (BeNeVol 2010), pages 80–84, Lille, 2010.

2. Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. By no means: A study on aggregating software metrics. In Giulio Concas, Massimiliano Di Penta, Ewan Tempero, and Hongyu Zhang, editors, *2nd International Workshop on Emerging Trends in Software Metrics* (WETSoM 2011), Honolulu, HI, USA, 2011.

3. Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics. *International Conference on Software Maintenance* (ICSM 2011), IEEE, Williamsburg, VA, USA, 2011, *accepted*.

4. Karine Mordal, Nicolas Anquetil, Jannik Laval, Alexander Serebrenik, Bogdan Vasilescu, and Stéphane Ducasse. Practical software quality metrics aggregation. Under review for the *Journal of Software Maintenance and Evolution* special issue of the *15th European Conference on Software Maintenance and Reengineering* (CSMR 2011), Wiley.

Although this thesis is written as a linear document, the actual research evolved from a pilot study and involved significant exploration of ideas, experimenting and back-tracking. The above publications relate to this thesis as follows.

The two early papers helped set the grounds for the work presented in this thesis. Specifically, the BeNeVol'10 contribution (paper 1) represents a pilot theoretical and empirical comparative study of different aggregation techniques for software metrics (the *mean*, as well as the *Gini*, *Theil*, *Atkinson*, and *Kolm* inequality indices). In this paper we aggregated *source lines of code* (SLOC) values from class to package level, and we studied correlation between the aggregated values and the *number of defects per package*. As a result on a single snapshot of ArgoUML, we observed that the choice of aggregation technique matters, i.e., it influences the correlation between the aggregated SLOC values and the validation metric (*number of defects per package*). Furthermore, we observed that the aggregation techniques fall into two groups, with high and statistically significant correlation among the

techniques in each group, i.e., they convey the same information: the *mean* and the *Kolm* index on the one hand, and the *Gini*, *Theil*, and *Atkinson* indices on the other hand.

However, a significant threat to the validity of such studies is the representativeness of the case study considered. Consequently, in paper 2, published at WETSoM'11, we investigated three case studies (ArgoUML, Adempiere, and Mogwai Java Tools) by means of the aforementioned aggregation techniques augmented by the *sum*, *median*, and *Hoover* inequality index. We observed that indeed the choice of aggregation technique influences the correlation with defects. However, the separation of the techniques into two groups with high and statistically significant correlation among the elements in each group was not as clear as before, and was not consistent across the systems. Nevertheless, the *Gini*, *Theil*, *Atkinson*, and *Hoover* inequality indices showed high and statistically significant correlation among themselves, i.e., the aggregated values obtained using these techniques convey the same information.

In paper 3, to appear at ICSM'11, we built on the BeNeVol'10 and WETSoM'11 articles, and we extended them in two ways. First, we presented the results of an extensive correlation study of the most widely-used traditional (*mean*, *median*, *sum*, *standard deviation*, *variance*, *skewness*, and *kurtosis*) and econometric (*Gini*, *Theil*, *Atkinson*, *Hoover*, and *Kolm*) aggregation techniques, applied to lifting SLOC values from class to package level in the 106 systems comprising the Qualitas Corpus [110]. Second, apart from measuring the strength of the correlation between the various aggregation techniques, we also investigated the nature of this relation, and studied its evolution on a subset of 12 systems from the Qualitas Corpus. As a result, we consistently observed a clear separation of the techniques into two groups (the *mean* and the *Kolm* index on the one hand, and the *Gini*, *Theil*, *Hoover*, and *Atkinson* indices on the other hand), with high and statistically significant correlation among the elements in each group. In spite of this high correlation, one index or another might still be preferred, and we discussed some of the rationale behind such a choice.

Finally, in the JSME submission (paper 4) we investigated the relation between the Squale model and the econometric inequality indices, and we showed that although Squale has been designed for software metrics, it shares common properties with the inequality indices. Additionally, since one of the strong points of Squale is its sensitivity to bad (unwanted) values, we performed an empirical comparison with data based on metrics extracted from Eclipse[1], in which we tested the behavior of the Squale aggregation and the econometrical inequality indices in presence of an increasingly larger number of low individual marks.

[1] `http://www.eclipse.org/`

Part I

Theoretical analysis

Chapter 2

Traditional aggregation techniques

As mentioned earlier, *aggregation of software metrics* can be understood in two ways.

First, there is a need to combine different metrics as recommended by quality-model design methods such as Factor-Criteria-Metric (FCM) [75], or Goal-Question-Metric (GQM) [13], i.e., aggregation is performed on values obtained by applying *different metrics* to the *same software artifacts*. For example, *cyclomatic complexity* might be combined with *test coverage* metrics to stress the importance to cover complex methods rather than simple accessors [74].

Second, there is a need to obtain insights in the quality of an entire system based on the metric values obtained from low-level system elements, i.e., aggregation is performed on values obtained by applying the *same metric* to *different software artifacts*. Examples thereof include the *Weighted Methods per Class* (WMC) or *average number of lines of code* metrics, as discussed in the Introduction. Additionally, using the FCM model in [70] to assess the maintainability of a system involves computation of such metrics as *number of source lines of code* (SLOC), *cyclomatic complexity*, *number of methods per class*, or *inheritance depth* (DIT). All these metrics can only be computed for methods and/or classes. However, the maintainability assessment requires insights at system level. One possible approach is to first aggregate each metric from method/class level to the system level, and then combine these system-level results into a unified assessment.

In this thesis we focus on the latter, i.e., aggregation performed on values obtained by applying the *same metric* to *different software artifacts*. In this sense, we study three categories of aggregation techniques: standard summary statistics (e.g., *mean*, *median*, etc.), econometric inequality indices (e.g., *Gini*, *Theil*, etc.), and threshold-based approaches (e.g., the aggregation proposed in the *Squale* quality model [76]). In this chapter we

detail the traditional aggregation techniques, and discuss their appropriateness for software metrics. Later, we discuss econometric inequality indices as aggregation techniques in Chapter 3, and threshold-based approaches to aggregation of software metrics in Chapter 4.

Throughout the current and the following two chapters we will denote as $\mathcal{X}$ the collection $\{x_1, \ldots, x_n\}$ of values to be aggregated. An element $x_i \in \mathcal{X}$ is either the value of a certain *metric* for class or method i, or the *income* received by individual i, when we refer to the econometric inequality indices in Chapter 3.

In order to aggregate values obtained by applying the same metric to different software artifacts, standard summary statistics such as additive measures (*sum*), central tendency measures (*mean*, *median*), statistical dispersion measures (*standard deviation*, *variance*), or distribution shape measures (*skewness*, *kurtosis*) are often used [12, 56, 64].

The ***sum***, denoted as x_{total}, is probably the simplest aggregation measure. The sum is defined as

$$x_{total} = \sum_{i=1}^{n} x_i. \tag{2.1}$$

However, albeit its simplicity, the *sum* loses the absolute value information (recall, e.g., that MI [83] and *code delta* [47] may have negative values) and it is unbounded, making relative comparisons difficult.

The arithmetic ***mean*** (or simple average), denoted as $\bar{x}$, is defined as

$$\bar{x} = \frac{x_{total}}{n}. \tag{2.2}$$

The *mean* is a good measure of centrality, but only for symmetrical (e.g., normal) distributions. The mean can be misleading for skewed distributions due to influence of outliers, since it does not convey the standard deviation of the population and may dilute unwanted values in the generally-acceptable results [55].

To illustrate this problem, consider the example of the maintainability index MI, whose definition typically involves computing the *average number of lines of code per module* [83], such that systems with lower average LOC are preferred. Table 2.1 presents the *number of lines of code* values for four classes in two consecutive versions of a software system. When computing *mean*(SLOC) in both cases, *Version 2* appears to be better since it has a lower average value. However, this hides the fact that *Class A* became an outlier after the refactoring. Therefore, while the aggregated value is better (i.e., smaller), the maintainability of the system might have actually decreased.

Table 2.1: SLOC values for four classes in two consecutive versions of a system.

Class	A	B	C	D	*Mean*
Version 1	24	25	27	24	25.0
Version 2	71	9	10	8	24.5

An alternative approach to highlighting undesirable values is to increase their weight in the average. However, as shown in [76], this suffers from similar drawbacks, thus incorrectly suggesting a decrease of the software quality while the code has actually improved, or vice versa.

The ***median***, denoted as $\tilde{x}$, is defined as

$$\tilde{x} = \begin{cases} x_{(n+1)/2}, & \text{if } n \text{ is odd;} \\ \frac{1}{2}(x_{n/2} + x_{n/2+1}), & \text{if } n \text{ is even.} \end{cases} \tag{2.3}$$

The *median* is less sensitive to outliers than the mean, but can yield different results if a small change occurs in the data set, e.g., one value is removed:

$$\begin{aligned} \tilde{x}(1,1,1,13,13,13) = \bar{x}(1,1,1,13,13,13) &= 7 \\ \tilde{x}(1,1,13,13,13) &= 13 \\ \bar{x}(1,1,13,13,13) &= 8.2 \end{aligned}$$

The *variance* and *standard deviation* are measures of statistical dispersion. The ***variance***, denoted as *var*, is defined as

$$var(\mathcal{X}) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{2.4}$$

On the other hand, the ***standard deviation***, denoted as $\sigma(\mathcal{X})$, is equal to the square root of the variance.

$$\sigma(\mathcal{X}) = \sqrt{var(\mathcal{X})} \tag{2.5}$$

Both the standard deviation and the variance are based on the mean, hence they also become unreliable for highly skewed distributions, where they do not convey information about the asymmetry.

The skewness and the kurtosis offer two more alternatives to aggregation of software metrics. ***Skewness***, denoted as γ_1, measures the asymmetry of a distribution, and is defined as

$$\gamma_1(\mathcal{X}) = \frac{1}{n} \frac{\sum_{i=1}^{n} (x_i - \bar{x})^3}{\sigma^3(\mathcal{X})}. \tag{2.6}$$

A distribution with relatively few low values typically has negative skew, and is said to be *left-skewed*. On the other hand, a distribution with relatively few high values has positive skew, and is said to be *right-skewed*. Skew is zero when the distribution is symmetric about the mean.

In contrast, ***kurtosis***, denoted as γ_2, measures the peakedness of a distribution, and is defined [105] as

$$\gamma_2(\mathcal{X}) = \frac{1}{n} \frac{\sum_{i=1}^{n} (x_i - \bar{x})^4}{\sigma^4(\mathcal{X})} \tag{2.7}$$

For the normal distribution, $\gamma_2 = 3$. For this reason, kurtosis is sometimes defined [105] as $\gamma_2'(\mathcal{X}) = \gamma_2(\mathcal{X}) - 3$, which is also known as *excess kurtosis*. High kurtosis corresponds to a distribution with sharp peaks and long fat tails, while low kurtosis corresponds to a distribution with rounded peaks and short thin tails. A distribution with $\gamma_2 > 3$ ($\gamma_2' > 0$) is called *leptokurtic*, one with $\gamma_2 < 3$ ($\gamma_2' < 0$) is called *platykurtic*, and one with $\gamma_2 = 3$ ($\gamma_2' = 0$) is called *mesokurtic*.

Similar to the variance and standard deviation, the skewness and kurtosis are unbounded, hence cause difficulties when comparing systems with different population sizes [113].

Software metrics such as SLOC are typically right-skewed and leptokurtic. For example, for the distribution of SLOC per class in Hibernate release 3.6.0-beta4 (Figure 2.1), skewness = 12.6 and kurtosis = 245.5.

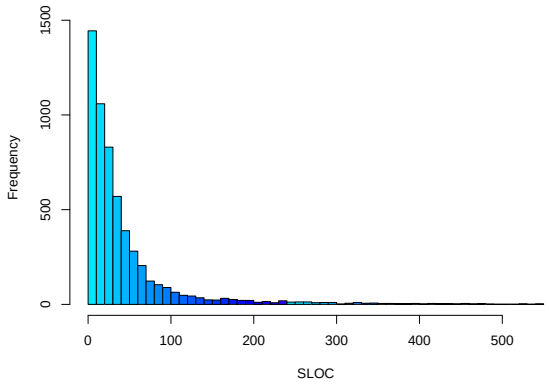

Figure 2.1: Distribution of SLOC in Hibernate in release 3.6.0-beta4.

Chapter 3

Econometric aggregation techniques

The second category of aggregation techniques considered consists of econometric inequality indices, commonly used to study inequality of income or welfare distributions [29–31]. Specifically, we consider the *Gini* [43], *Theil* [111], *Atkinson* [8], *Hoover* [54] (also known as the Ricci-Schutz coefficient, the Robin Hood index, or the relative mean deviation), and *Kolm* [60] income inequality indices. When the distribution is equal, i.e., each person has the same share of the overall income or wealth, any such measure reaches its absolute minimum, zero. Deviations from this equality, when some individuals assume a higher share of the total income or wealth than others, correspond to an increase in the value of the inequality measure.

In contrast with the traditional techniques, the econometric inequality indices can be used successfully to aggregate software metrics, since they provide a synthesis of the skewness, kurtosis, mean, and variance statistics of the data, without being affected by skewed distributions [113]. In this chapter we introduce and define the inequality indices considered. Later, in Section 3.2, we discuss more advanced mathematical properties of the inequality indices, as well as implications of these properties on aggregation of software metrics.

3.1 Inequality indices and software metrics

In econometrics, such indices are used to measure and to explain the inequality of income or wealth distributions. For example, inequality indices applied to 138 countries show reductions in global inequality of gross domestic product (GDP) per capita during the 1980s and 1990s [94].

Similar insights can be derived by applying inequality indices to software metrics. For example, one of the pieces of information often required when analyzing a software system is the degree of concentration of function-

ality within it [113]. In this sense, one wishes to understand, e.g., whether functionality is widely distributed across classes (corresponding to *low inequality* of income in econometrics), or whether there are only a few classes implementing most of the functionality in the system (corresponding to *high inequality* of income in econometrics). Obtaining such information for a certain version of a system can help indicate the presence of god classes or machine-generated code, while tracking it across multiple releases can help reveal significant architectural shifts [99, 113].

The intuition above can be formalized using the Lorenz curve, initially proposed to measure and visualize the concentration of wealth [66]. Consequently, a Lorenz curve plots on the y-axis the percentage of wealth cumulatively assumed by the bottom $x\%$ of the population. It ranges between the diagonal $y = x$ (i.e., the line of *perfect equality*), corresponding to each individual assuming the same percentage of the total wealth, and the line that is zero for all values of $x < 1$ and 1 for $x = 1$ (i.e., the line of *perfect inequality*), corresponding to a single individual assuming the total wealth.

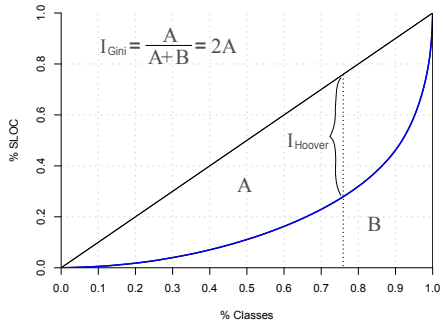

Figure 3.1: Lorenz curve for SLOC in Hibernate in release 3.6.0-beta4.

The Lorenz curve can be used to study the concentration of functionality within a software system. Figure 3.1 shows an example of a Lorenz curve for the *number of source lines of code* (SLOC) metric in Hibernate release 3.6.0-beta4. We observe significant deviation from the diagonal, i.e., there is high inequality in the distribution of SLOC among the classes in Hibernate. Additionally, we observe that approximately 70% of all classes accumulate only 20% of the total size of Hibernate. Similarly, the top 10% of all classes accumulate more than 50% of the total size. This finding is in line with the Pareto principle [84], stating that roughly 80% of the effects stem from 20% of the causes. Evidence for the Pareto principle have been reported in numerous areas of human activity, including computer science. For example, commit, email, and bug reporting activities in open-source projects have been reported to adhere to the Pareto principle [46]. Similarly, 20% of the most reported bugs in Microsoft Windows and Office are known to cause 80% of the errors [93].

Two of the inequality indices considered in this thesis (the Gini index, I_{Gini}, and the Hoover index, I_{Hoover}) are derived from the Lorenz curve, and more effectively summarize it than our informal observations above.

First, if A is the area between the line of perfect equality and the Lorenz curve, and B is the area below the Lorenz curve, then the ***Gini index*** is computed as $\frac{A}{A+B}$ [33]. More formally, I_{Gini} can be defined to measure inequality between groups or between the individuals within a group.

For the population $\mathcal{X} = \{x_1, \ldots, x_n\}$, let $\mathcal{G}$ be a mutually exclusive and completely exhaustive (MECE) partitioning into m groups $\{G_1, G_2, \ldots, G_m\}$. Let w_j be the *income share* of group G_j, i.e., the ratio of the income of G_j and the total income.

$$w_j = \frac{\sum_{x \in G_j} x}{x_{total}} \tag{3.1}$$

Let p_j be the *population share* of group G_j, i.e., the ratio of the cardinality of G_j and the cardinality of $\mathcal{X}$.

$$p_j = \frac{|G_j|}{|\mathcal{X}|} = \frac{|G_j|}{n} \tag{3.2}$$

Consequently, the Gini index *between* groups can be expressed as the result of a comparison, across all groups, of the ratio between income and population shares [24]:

$$I_{\text{Gini}}(\mathcal{G}) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} p_i p_j \left| \frac{w_i}{p_i} - \frac{w_j}{p_j} \right|, \tag{3.3}$$

where $|x|$ is the absolute value of x.

Alternatively, rather than measuring the inequality between different groups of individuals, e.g., inequality of income between the European Union member states, or inequality of size between the different components of Hibernate (*engine*, *persister*, *testing*, etc.), one might be interested in measuring inequality *within* a certain population (group) of individuals. It is easy to observe that when each group consists of only one individual, $m = n$, $p_i = \frac{1}{n}$, and $w_i = \frac{x_i}{\sum_{x \in \mathcal{X}} x}$, and equation 3.3 rewrites to

$$I_{\text{Gini}}(\mathcal{X}) = \frac{1}{2n^2 \bar{x}} \sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j| \tag{3.4}$$

Second, the ***Hoover index*** [54] (also known as the Ricci-Schutz coefficient, the Robin Hood index, or the relative mean deviation) is equivalent to the maximum vertical distance between the Lorenz curve and the line of perfect equality (Figure 3.1). For the grouping $\mathcal{G}$, I_{Hoover} measures the

total absolute differences between the income and population shares of the groups, and is defined [4] as

$$I_{\text{Hoover}}(\mathcal{G}) = \frac{1}{2}\sum_{j=1}^{m} |w_j - p_j| \,, \tag{3.5}$$

where w_j is the income share of G_j and p_j is the population share of G_j.

Alternatively, at the individual level, equation 3.5 rewrites to

$$I_{\text{Hoover}}(\mathcal{X}) = \frac{1}{2n\bar{x}}\sum_{i=1}^{n} |x_i - \bar{x}| \,, \tag{3.6}$$

when each group consists of a single individual, for $m = n$, $p_j = \frac{1}{n}$, and $w_j = \frac{x_j}{\sum_{x \in \mathcal{X}} x}$.

To summarize, for a population $\mathcal{X}$ one can compute the inequality between the individuals $\{x_1, \ldots, x_n\}$ regardless of how $\mathcal{X}$ is partitioned (equations 3.4 or 3.6). Alternatively, given a partitioning $\mathcal{G} = \{G_1, \ldots, G_m\}$, thus the population and income shares of each group, one can compute the inequality between the groups $\{G_1, \ldots, G_m\}$ regardless of the actual dispersion of the distribution of income among individuals in each group (equations 3.3 or 3.5). However, when computing the inequality between all individuals $\{x_1, \ldots, x_n\}$, i.e., *within* the entire population, it is often desirable to assess the contribution to the total inequality introduced by the partitioning, i.e., the contribution of the inequality *between* the groups.

Such a property is desirable, e.g., when measuring the inequality of size (SLOC) between the classes in a software system which is organized into packages. In this sense, an important question in interpreting the inequality value aggregated on a system level pertains to the extent to which the result can be attributed to differences between system subcomponents. For example, one might be interested in assessing whether the disproportionately large classes are concentrated only in few packages, or they are spread over the entire system.

To answer this question it is therefore desirable to express the aggregation (inequality) result computed at a system level in terms of *between-group* and *within-group* components, hence obtain a clear separation of the inequality introduced by the partitioning into $\{G_1, \ldots, G_m\}$ from the inequality within each group G_j, $1 \leq j \leq m$. Various approaches to decomposition of a global-level inequality result into *between-group* and *within-group* contributions have been proposed for various inequality indices [16,17,30,85,101]. We refer to such a property of an inequality index as *decomposability*, and we discuss it in more detail in Section 3.2. Now, for the purpose of exposition, we restrict to the most common, additive decomposition of the aggregation result computed at a system level as the sum of a non-negative *within-group* term and a non-negative *between-group* term, where the *within-group* contribution is itself a weighted sum of applying the same aggregation technique

(inequality index) at the subcomponent level [101]. In econometrics, one commonly further requires that the sum of the weighting coefficients be 1.

$$\begin{aligned} I(\mathcal{X}) &= I^{within} + I^{between} \\ &= \sum_{j=1}^{m} \omega_j I(\mathcal{X}_j) + I^{between}, \end{aligned} \tag{3.7}$$

where $\mathcal{X}_j$ is the distribution of values in G_j, and ω_j is the weight assigned to the inequality within G_j in the total *within-group* term, $1 \leq j \leq m$.

Furthermore, the *between-group* term can be used to measure to what extent the aggregated value at the system level (i.e., the overall inequality) can be *explained* by a specific partitioning of the system into subcomponents [29, 99]. We discuss explanation of inequality in more detail in Section 3.2. The lack of such a decomposability property is a major shortcoming of the Gini and Hoover inequality indices [28].

The ***Theil index*** [111] is a member of the *generalized entropy class* of inequality measures and it satisfies the decomposability property. For a grouping $\mathcal{G}$, the Theil index can be defined to measure inequality between groups [111] as

$$I_{\text{Theil}}(\mathcal{G}) = \sum_{j=1}^{m} \left(w_j \log \frac{w_j}{p_j} \right), \tag{3.8}$$

where w_j is the *income share* of G_j, and p_j is the *population share* of G_j.

Similarly, it is easy to observe that when each group consists of only one individual, $m = n$, $p_j = \frac{1}{n}$, and $w_j = \frac{x_j}{\sum_{x \in \mathcal{X}} x}$. Hence, the Theil index can be defined to measure inequality between the individuals within a group as

$$I_{\text{Theil}}(\mathcal{X}) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{x_i}{\bar{x}} \log \frac{x_i}{\bar{x}} \right) \tag{3.9}$$

Pertaining to decomposability (equation 3.7), for a MECE partitioning $G_1, \ldots, G_m$, I_{Theil} can be decomposed into a *between-group* and a *within-group* component, as follows.

$$I_{\text{Theil}}(\mathcal{X}) = I_{\text{Theil}}^{between} + I_{\text{Theil}}^{within}, \tag{3.10}$$

such that

$$I_{\text{Theil}}^{within} = \sum_{j=1}^{m} w_j I_{\text{Theil}}(G_j), \tag{3.11}$$

where w_j is the income share of G_j, and $I_{\text{Theil}}(G_j)$ is computed using equation 3.9.

The *between-group* component $I_{\text{Theil}}^{between}$ can be computed either using equation 3.8, or, by further exploiting the decomposability property of I_{Theil}, using equation 3.9 and replacing each $x \in \mathcal{X}$ with $\bar{x}_j$, where $\bar{x}_j$ is the mean of the group G_j that x is part of, i.e., $\bar{x}_j = \frac{1}{|G_j|} \sum_{x \in G_i} x$, $1 \leq j \leq m$ [101].

$$I_{\text{Theil}}^{between} = I_{\text{Theil}}(\underbrace{\bar{x}_1, \bar{x}_1, \ldots, \bar{x}_1}_{|G_1|}, \underbrace{\bar{x}_2, \bar{x}_2, \ldots, \bar{x}_2}_{|G_2|}, \ldots, \underbrace{\bar{x}_m, \bar{x}_m, \ldots, \bar{x}_m}_{|G_m|}) \quad (3.12)$$

Conducive to illustrating decomposability and to giving a more intuitive interpretation to the Theil index, we measure inequality of SLOC per class in JMoney release 0.4.4, a personal finance (accounting) manager written in Java, taking into account the organization of JMoney into four packages.

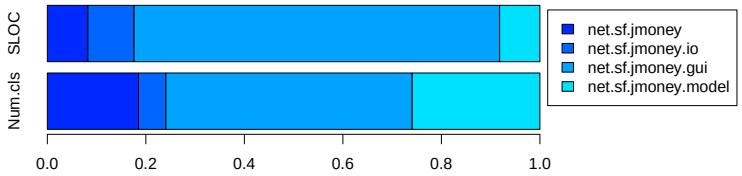

Figure 3.2: Inequality of SLOC: population (classes) and income (SLOC) shares for each package in JMoney release 0.4.4.

Figure 3.2 plots the population and income shares for each package in JMoney. We observe, e.g., that the *net.sf.jmoney.gui* package which implements the graphical user interface contains 50% of the classes, which account for approximately 75% of the total size of JMoney.

To characterize the inequality of SLOC *between* packages, we apply I_{Theil} from equation 3.8. The total $I_{\text{Theil}}^{between}$ is obtained by summation of all values in the last column of Table 3.1, which provides a complete overview of the population and income shares of each package, as well as the contribution of each package to the sum (the total $I_{\text{Theil}}^{between}$). Alternatively, $I_{\text{Theil}}^{between}$ can be computed using equation 3.9 by replacing each $x \in G_j$ with $\bar{x}_j$, where $\bar{x}_j$ is the mean of G_j [101], $j = 1, \ldots, 4$.

Table 3.1: Computing $I_{\text{Theil}}^{between}$ for the four packages in JMoney 0.4.4.

Package	Num.classes	SLOC	p_j	w_j	$I_{\text{Theil}}^{between}$
- *net.sf.jmoney*	10	634	0.19	0.08	-0.07
- *net.sf.jmoney.io*	3	721	0.05	0.09	0.05
- *net.sf.jmoney.gui*	27	5714	0.50	0.75	0.29
- *net.sf.jmoney.model*	14	630	0.26	0.08	-0.09
Total	54	7699	1	1	0.18

So far we were able to compute $I_{\text{Theil}}^{between}$ between the four packages in JMoney 0.4.4. In order to capture also the dispersion of the distribution of SLOC values among classes in each package (Figure 3.3), a natural way to move forward and compute $I_{\text{Theil}}^{within}$ for each package is to apply the same procedure at the aggregation level of a package rather than the entire system, using equation 3.9.

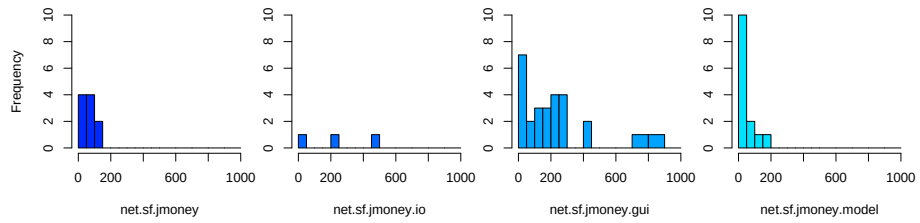

Figure 3.3: Distribution of SLOC for the packages in JMoney release 0.4.4. The colors match the ones used in Figure 3.2

Table 3.2 presents the results of computing $I_{\text{Theil}}^{within}$ for each of the four packages in JMoney release 0.4.4. The total $I_{\text{Theil}}^{within}$ is obtained by weighted summation of all values in the last column of Table 3.2, using the income shares as weights (equation 3.11).

Table 3.2: Computing $I_{\text{Theil}}^{within}$ for each of the four packages in JMoney 0.4.4.

Package	Num.classes	SLOC	p_j	w_j	$I_{\text{Theil}}^{within}$
- *net.sf.jmoney*	10	634	0.19	0.08	0.17
- *net.sf.jmoney.io*	3	721	0.05	0.09	0.44
- *net.sf.jmoney.gui*	27	5714	0.50	0.75	0.38
- *net.sf.jmoney.model*	14	630	0.26	0.08	0.60
Total	54	7699	1	1	0.39

We note that the result obtained by summation of the total $I_{\text{Theil}}^{between}$ and $I_{\text{Theil}}^{within}$ previously computed is equal to the value of I_{Theil} applied to the union of the four sets of SLOC values per class corresponding to each of the four packages (equation 3.10).

$$I_{\text{Theil}}(\text{JMoney}) = I_{\text{Theil}}^{between} + I_{\text{Theil}}^{within} = 0.18 + 0.39 = 0.57,$$

which corresponds to low inequality when compared to the maximal value of I_{Theil} for a population of 54 classes, i.e., $\log 54 \simeq 3.99$. We discuss interpretation of inequality index results in more detail in Section 3.2.

In addition to I_{Theil} above, also known as the first Theil index, Theil [111] has also introduced the second Theil index, known as the ***mean logarithmic deviation***, and defined as

$$I_{\text{MLD}}(\mathcal{X}) = \frac{1}{n}\sum_{i=1}^{n}\left(\log\frac{\bar{x}}{x_i}\right) \tag{3.13}$$

I_{MLD} is another member of the *generalized entropy class* of inequality measures, and it also satisfies the *decomposability* property [29].

Moreover, we study the Atkinson and the Kolm inequality indices. The ***Atkinson index*** [8], denoted as I_{Atkinson}, is defined as

$$I^{\alpha}_{\text{Atkinson}}(\mathcal{X}) = 1 - \frac{1}{\bar{x}}\left(\frac{1}{n}\sum_{i=1}^{n}x_i^{1-\alpha}\right)^{1/(1-\alpha)} \tag{3.14}$$

In our implementation, I_{Atkinson} is a standard instantiation of the Atkinson family of indices, for $\alpha = 0.5$ [30, 120], i.e.,

$$I_{\text{Atkinson}}(\mathcal{X}) = 1 - \frac{1}{\bar{x}}\left(\frac{1}{n}\sum_{i=1}^{n}\sqrt{x_i}\right)^{2} \tag{3.15}$$

Finally, the ***Kolm index*** [60], denoted as I_{Kolm}, is defined as

$$I^{\beta}_{\text{Kolm}}(\mathcal{X}) = \frac{1}{\beta}\log\left[\frac{1}{n}\sum_{i=1}^{n}e^{\beta(\bar{x}-x_i)}\right] \tag{3.16}$$

In our implementation, I_{Kolm} is a standard instantiation of the Kolm family of indices, for $\beta = 1$ [120], i.e.,

$$I_{\text{Kolm}}(\mathcal{X}) = \log\left[\frac{1}{n}\sum_{i=1}^{n}e^{\bar{x}-x_i}\right] \tag{3.17}$$

In the remainder of this thesis we refer to the standard instantiations of the Atkinson and Kolm inequality indices, unless stated otherwise.

3.2 Mathematical properties

In this section we discuss a number of properties of the inequality indices, as well as implications of these properties on aggregation of software metrics.

3.2.1 Domain

The domain of the aggregation technique determines its applicability to classes of software metrics. Econometric inequality indices are usually applied to income or welfare distributions, i.e., to sets of positive values. However, there are software metrics, such as the *Maintainability Index* MI [83], that may have negative values. Since $\log x$ and $\sqrt{x}$ are undefined for $x < 0$, I_{Theil}, I_{MLD}, and I_{Atkinson} are undefined as well. Unlike these indices, I_{Gini}, I_{Hoover}, and I_{Kolm} can be used to aggregate negative values.

Moreover, as $\log 0$ is undefined, direct application of the Theil index formula in presence of zero values (equation 3.9) is not possible. The usual approach in econometrics is to consider that a person with no income does not contribute to the income distribution, hence $I_{\text{Theil}}(x_1, \ldots, x_{n-1}, 0)$ should be defined as $I_{\text{Theil}}(x_1, \ldots, x_{n-1})$. While this approach is valid for software metrics where zero denotes emptiness (e.g., SLOC, *number of classes in a package*, etc.), it is less appropriate, e.g., for the *normalized distance from the main sequence* D_n [72], where $D_n = 0$ is the most desirable situation, indicating perfect balance between abstractness and instability.

Alternatively, for the sake of simplicity, one can replace 0 by a very small $\varepsilon > 0$, such that $I_{\text{Theil}}(x_1, \ldots, x_{n-1}, 0) \stackrel{\text{def}}{=} I_{\text{Theil}}(x_1, \ldots, x_{n-1}, \varepsilon)$. This observation can be generalized for an arbitrary number of zeros as long as at least one non-zero value is present.

Finally, formulas for the Gini index (equation 3.4), the Theil index (equation 3.9), and the Atkinson index (equation 3.15) involve divisions by $\bar{x} = \frac{x_{total}}{n}$, while the formula for the Hoover index (equation 3.6) involves division by x_{total}. Hence, these indices are undefined if $x_{total} = 0$.

3.2.2 Interpretation

The interpretation of the aggregated value depends on the range of the aggregation technique. For example, 0.99 indicates a very high degree of inequality if I_{Gini}, I_{Hoover}, or I_{Atkinson} are considered, while in case of I_{Theil} the interpretation would depend on the number of values being aggregated. All the indices considered are zero when all individuals have identical income.

The Gini, Hoover, and Atkinson indices range over $\left[0, 1 - \frac{1}{n}\right]$ if all the values being aggregated are positive [4, 90]. In general, this is not necessarily the case, e.g., $I_{\text{Gini}}(1, -1.5) = -2.5$ and $I_{\text{Hoover}}(1, -1.5) = -2.5$. As mentioned in the previous paragraph, $I_{\text{Atkinson}}(1, -1.5)$ is undefined.

In contrast, the Theil index ranges over $[0, \log n]$ [4]. To facilitate direct comparison with the Gini, Hoover, and Atkinson indices, I_{Theil} can be normalized to the range $[0, 1]$, e.g., by dividing it by $\log n$, or by computing $e^{-I_{\text{Theil}}}$. The Kolm index ranges over non-negative reals [60].

3.2.3 Invariance

We call the aggregation technique *invariant with respect to addition* if

$$I(x_1, \ldots, x_n) = I(x_1 + c, \ldots, x_n + c), \tag{3.18}$$

for any $x_1, \ldots, x_n$ and c, provided that $I(x_1 + c, \ldots, x_n + c)$ exists.

Similarly, we call the aggregation technique *invariant with respect to multiplication* (scale invariant) if

$$I(x_1, \ldots, x_n) = I(x_1 \cdot c, \ldots, x_n \cdot c), \tag{3.19}$$

for any $x_1, \ldots, x_n$ and c, provided that $I(x_1 \cdot c, \ldots, x_n \cdot c)$ exists.

It can be shown that I_{Gini}, I_{Theil}, I_{Hoover} and I_{Atkinson} are invariant with respect to multiplication [4, 8, 29]. Unlike these indices, I_{Kolm} is invariant with respect to addition [60].

In econometrics, an important argument in favor of scale invariance is the independence from the unit in which a variable is measured, e.g., it is not necessary to deal with currency conversions when measuring the inequality of income measured in USD or EUR.

Similar benefits arise when applying scale-invariant measures to software metrics. For example, in case of aggregating *source lines of code* (SLOC) measured per file, the results obtained are not affected if percentages of the total SLOC are considered rather than the SLOC values themselves.

Additionally, scale invariance enables that inequality be comparable across different metrics. For example, one might be interested in comparing the inequality in *lines of code* to that in *percentage of comments* per file across a given system, which is otherwise meaningless. We illustrate such a comparison in Figure 3.4, which displays the Lorenz curves for the *source lines of code* (SLOC) and *percentage of lines with comments* (PLwC) metrics in ArgoUML in release 0.30.2. We observe significant deviation from the diagonal for L(SLOC) and little deviation from the diagonal for L(PLwC), i.e., across the classes in ArgoUML there is high inequality in the distribution of SLOC, and low inequality in the distribution of PLwC. These observations are confirmed by applying the Gini and the Hoover indices: $I_{\text{Gini}}(\text{SLOC}) = 0.69$, $I_{\text{Gini}}(\text{PLwC}) = 0.19$; $I_{\text{Hoover}}(\text{SLOC}) = 0.53$, $I_{\text{Hoover}}(\text{PLwC}) = 0.14$.

However, by multiplying each value by a constant, it is less clear that the inequality actually remained unchanged, since the *rich* benefit more than the *poor* in absolute terms, e.g., from an increase by 10% in everyone's income. This drawback becomes much clearer when considering also that

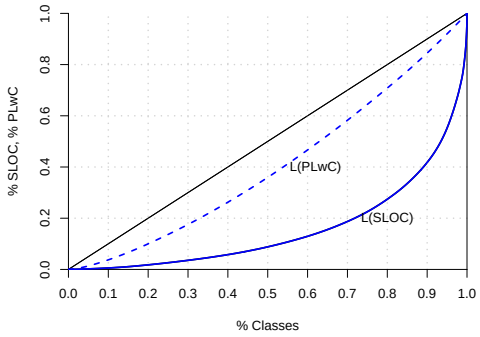

Figure 3.4: Lorenz curves for SLOC and PLwC in ArgoUML 0.30.2.

the inequality indices are zero when all individuals have identical income. For example, in case of aggregating *cyclomatic complexity* measured per class, a Java package in which all classes have equally low complexity is probably much more desirable than one in which all classes have equally high complexity.

On the other hand, the invariance with respect to addition allows to ignore, e.g., headers containing the licensing information and included in all source files.

3.2.4 Symmetry

For an aggregation technique, *symmetry* [40] or *impartiality* [60] states that the final result should not be dependent on the order of the elements being aggregated, i.e., it should be invariant to permutations of the aggregated values.

Symmetry is very important for software metrics since one does not want, e.g., the aggregated value to depend on whether the name of one of the files has been changed from "AAAA" to "ZZZZ". All inequality indices are symmetric [60].

3.2.5 Transfers principle

According to the Pigou-Dalton principle of transfers [97], a strictly positive transfer from a richer to a poorer individual, without reversing their status and with leaving all other incomes unchanged, should result in a strictly positive reduction in the inequality index.

I_{Hoover} is not affected by transfers between people on the same side of the mean [8], hence it is said not to satisfy the principle of transfers.

$$I_{\text{Hoover}}(1,1,3,30) = I_{\text{Hoover}}(1,2,2,30) \simeq 0.61$$

In contrast, I_{Gini}, I_{Theil}, I_{Atkinson}, I_{Kolm}, and I_{MLD} all satisfy the principle of transfers [4, 60].

3.2.6 Sensitivity to transfers

Consider again the population $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, where x_i is the income of individual i. Assume, without loss of generality, that $\mathcal{X}$ is sorted. Assume a transfer of k euro occurs from a poorer person with income x_i to a richer person with income x_j, where $x_i \leq x_j$, without modifying any of the other incomes.

Among the inequality indices that satisfy the principle of transfers, there are differences in their relative sensitivity to transfers at different income levels. For example, in econometrics one is interested in assessing whether a transfer of 50 euro from a person earning 100 euro to a person earning 150 euro has the same impact as a transfer of 50 euro from a person earning 10000 euro to a person earning 10001 euro. Similarly, with software metrics, one might be interested in assessing how differently a decrease in SLOC of 20 for one class with SLOC 50 at the cost of an equivalent increase in SLOC for another class with SLOC 60 affects the overall inequality of SLOC than a decrease in SLOC of 20 for one class with SLOC 500 at the cost of an equivalent increase in SLOC for another class with SLOC 600.

Typically, one can distinguish between three *dimensions of inequality* [21]: inequality due to extreme relative wealth, inequality among the less extreme incomes, or inequality due to extreme poverty. In case of SLOC, extreme relative wealth, i.e., inequality associated with the *exceptionally rich*, corresponds to a non-egalitarian distribution of functionality caused by systems having few very big or complex components and many small or simple ones. Analogously, inequality due to *extreme poverty* is caused by systems having few very small components rather than few very big ones. Inequality *among the less extreme incomes* corresponds to a more uniform distribution of differences in functionality among the components of the system.

A change in I_{Gini} depends on the number of individuals with incomes in between x_i and x_j, i.e., on the ranks (i,j) of the individuals rather than their income [4]. Therefore, for a normal distribution I_{Gini} is sensitive to transfers around the middle (*among the less extreme incomes*) [4,21]. Similarly, for a positively-skewed distribution (e.g., SLOC) I_{Gini} is sensitive to transfers at the low (poor) end, since for such distributions there are many more values in the $[20, 50]$ interval than, e.g., in the $[320, 350]$ interval. It follows that a transfer from one method/class with SLOC 20 to another with SLOC 50 will have more effect on I_{Gini} than an equal transfer from one method/class with SLOC 320 to another with SLOC 350.

In contrast, a change in I_{Theil} or I_{MLD} depends on the ratio of x_i and x_j. It follows that the lower the income level (*extreme poverty*), the more sensitive to transfers I_{Theil} and I_{MLD} are [4]. In contrast to the theoretical approach in [4], Champernowne [21] reports opposite findings, i.e., that I_{Theil} is sensitive to inequality due to the exceptionally rich rather than the

extremely poor, after experimenting with a number of simulated distributions.

The sensitivity of $I^{\alpha}_{\text{Atkinson}}$ and I^{β}_{Kolm} to different dimensions of inequality depends on the values of the parameters α and β [60]. As α increases, $I^{\alpha}_{\text{Atkinson}}$ is more sensitive to transfers at the lower end of the distribution and less sensitive to transfers at the top. In the limiting case $\alpha \to \infty$, $I^{\infty}_{\text{Atkinson}}$ is only sensitive to transfers to or from the poorest individual. On the other hand, in the limiting case $\alpha = 0$, $I^{0}_{\text{Atkinson}} = 0$. Similarly, if $\beta = 0$ then $I^{0}_{\text{Kolm}} = 0$, and if $\beta \to \infty$ then I^{∞}_{Kolm} is only sensitive to transfers to or from the poorest individual.

3.2.7 Population principle

The population principle [29] states that a distribution $\mathcal{X}$ should be regarded equivalent from an inequality point of view to a distribution $\mathcal{X}'$ obtained by replicating $\mathcal{X}$ a number of times.

All inequality indices satisfy the population principle [29, 60]. Note that even traditional aggregation techniques such as the *mean* satisfy the population principle.

3.2.8 Decomposability

Decomposability into *between-group* and *within-group* components is the key property necessary for explanation of inequality by partitioning the values to be aggregated into disjoint groups. In econometrics such groups correspond, e.g., to education level, gender or ethnicity, while in software evolution research, e.g., to package, programming language and maintainer's name [99].

To explain inequality Cowell and Jenkins introduced the R index in [30], defined as the ratio of the inequality between the groups and the total amount of inequality, given a decomposable inequality index I and a MECE partitioning $\mathcal{G} = \{G_1, \ldots, G_m\}$.

$$R(\mathcal{G}) = \frac{I^{between}(\mathcal{G})}{I(\mathcal{X})} \tag{3.20}$$

R indicates what share of the inequality can be explained by the partitioning into $\{G_1, \ldots, G_m\}$, and it ranges between 0 and 1. $R = 0$ in case of a trivial partition of the population into one group, i.e., inequality can be completely attributed to inequality within the group. $R = 1$ corresponds to the case when the partition is "complete", i.e., every element of the population is considered as a group in itself.

For example, using R and I_{Theil}, in [1] expenditure in Indonesian households has been shown to be better explained by the education level of the head of the household (32.6%) than by the province of residence (18.9%) or by the gender of the household's head (2.6%). This suggests that, e.g., in

order to reduce expenditure inequality, the Indonesian government should invest in education rather than in reducing gender inequality. Similarly, it has been observed that 17.4% of inequality in file sizes (SLOC) of the Linux Debian *lenny* distribution can be explained by the package these files belong to, while the implementation language can account for only 5.32% of the inequality [99].

Alternative approaches to decomposability

Application of the R index requires that the inequality measure be decomposable into *between-group* and *within-group* components. Expressing the *within-group* component is straightforward for any inequality index, since the index itself is defined for populations of arbitrary size, i.e., selection of a measure for the entire population automatically provides a measure for any subgroup. On the other hand, there are two main approaches to expressing the *between-group* component in the literature [30]. Let $\mathcal{G} = \{G_1, \ldots, G_m\}$ be a given MECE partitioning of the population $\mathcal{X}$. Let n_j be the cardinality of G_j, for all j, $1 \leq j \leq m$.

First, the *between-group* component can be interpreted as a function of the *group means*, i.e.,

$$I^{between} = \Phi(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m), \tag{3.21}$$

where $\bar{x}_j$ is the mean of G_j, i.e., $\bar{x}_j = \frac{1}{n_j} \sum_{x \in G_j} x$, for all j, $1 \leq j \leq m$.

In this approach, the aggregation result computed at a system level is typically decomposed [17, 101] as the sum of a non-negative *within-group* term and a non-negative *between-group* term, i.e.,

$$I(\mathcal{X}) = I^{within} + I^{between} \tag{3.22}$$

In constructing the *between-group* term, inequality within each subgroup is eliminated using the group means, hence *between-group* inequality is measured as the inequality which would be experienced by the entire population if each person in a subgroup would receive the subgroup mean income [16].

Examples of inequality indices thereof are I_{Theil} and I_{MLD}, for which the *between-group* and the *within-group* components are

$$I^{within}_{\text{Theil}} = \sum_{j=1}^{m} w_j I_{\text{Theil}}(G_j) \tag{3.23}$$

$$I^{between}_{\text{Theil}} = I_{\text{Theil}}(\underbrace{\bar{x}_1, \bar{x}_1, \ldots, \bar{x}_1}_{(n_1)}, \ \underbrace{\bar{x}_2, \bar{x}_2, \ldots, \bar{x}_2}_{(n_2)}, \ \ldots, \ \underbrace{\bar{x}_m, \bar{x}_m, \ldots, \bar{x}_m}_{(n_m)})$$

and,

$$I_{\text{MLD}}^{within} = \sum_{j=1}^{m} p_j I_{\text{MLD}}(G_j) \tag{3.24}$$

$$I_{\text{MLD}}^{between} = I_{\text{MLD}}(\underbrace{\bar{x}_1, \bar{x}_1, \ldots, \bar{x}_1}_{(n_1)}, \ \underbrace{\bar{x}_2, \bar{x}_2, \ldots, \bar{x}_2}_{(n_2)}, \ \ldots, \ \underbrace{\bar{x}_m, \bar{x}_m, \ldots, \bar{x}_m}_{(n_m)}),$$

respectively, where w_j is the income share of G_j, and p_j is the population share of G_j.

Second, the *between-group* component can be interpreted as a function of the *group equally-distributed equivalents*, i.e.,

$$I^{between} = \Phi(\xi_1, \xi_2, \ldots, \xi_m), \tag{3.25}$$

where ξ_j is the equally-distributed equivalent for G_j. In econometrics, the *equally-distributed-equivalent* income for the whole population [8, 59], ξ, is the income that, if given to each person, results in an income vector which is "equivalent" to the actual distribution as measured by a given social-evaluation function W. In econometrics, social-evaluation functions are commonly associated with utilitarianism [36] and can be used to capture the well-being of economically isolated individuals in a society [35]. Formally, given a social-evaluation function $W : \mathbb{R}^n \to \mathbb{R}$ and the distribution of income $x_1, \ldots, x_n$, then the equally-distributed-equivalent ξ is determined by $W(x_1, \ldots, x_n) = W(\xi, \ldots, \xi)$. The equally distributed equivalent ξ_j for a group G_j is defined analogously.

In this approach, instead of measuring *between-group* inequality as the overall inequality which would result if each person received his/her subgroup's mean income, $I^{between}$ is instead computed as the overall inequality which would result if each person received his/her subgroup's equally-distributed equivalent income.

Both I_{Atkinson} and I_{Kolm} are defined with respect to such a social-evaluation function W, which assumes a particular form depending on the inequality index considered [16]. The resulting decomposition of I_{Atkinson} into *within-group* and *between-group* components is

$$I_{\text{Atkinson}}(\mathcal{X}) = I_{\text{Atkinson}}^{within} + I_{\text{Atkinson}}^{between} - I_{\text{Atkinson}}^{within} I_{\text{Atkinson}}^{between}, \tag{3.26}$$

where

$$I_{\text{Atkinson}}^{within} = \sum_{j=1}^{m} w_j I_{\text{Atkinson}}(G_j) \tag{3.27}$$

$$I_{\text{Atkinson}}^{between} = I_{\text{Atkinson}}(\underbrace{\xi_1, \xi_1, \ldots, \xi_1}_{(n_1)}, \ \underbrace{\xi_2, \xi_2, \ldots, \xi_2}_{(n_2)}, \ \ldots, \ \underbrace{\xi_m, \xi_m, \ldots, \xi_m}_{(n_m)}),$$

In the case of $I^{\alpha}_{\text{Atkinson}}$, ξ_j is expressed as

$$\xi_j = \left(\frac{1}{n_j} \sum_{i=1}^{n_j} x_i^{1-\alpha} \right)^{\frac{1}{1-\alpha}} \tag{3.28}$$

In contrast, the resulting decomposition of I_{Kolm} into *within-group* and *between-group* components is

$$I_{\text{Kolm}} = I^{within}_{\text{Kolm}} + I^{between}_{\text{Kolm}}, \tag{3.29}$$

where

$$\begin{aligned} I^{within}_{\text{Kolm}} &= \sum_{j=1}^{m} p_j I_{\text{Kolm}}(G_j) \\ I^{between}_{\text{Kolm}} &= \sum_{j=1}^{m} p_j \xi_j - \xi \end{aligned} \tag{3.30}$$

In the case of I^{β}_{Kolm}, ξ is expressed as

$$\xi = -\frac{1}{\beta} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{-\beta x_i} \right) \tag{3.31}$$

Discussion of these notions in further detail goes beyond the scope of this thesis[1]. Therefore, we limit ourselves to emphasizing that all the different forms of decomposition into *between-group* and *within-group* components suggested for different inequality indices and discussed above are equally-suited for explaining the inequality using R (equation 3.20).

Root-cause analyses

Apart from explanation of inequality at partition level, decomposability (via the R index) enables *root-cause analyses*, i.e., it provides means to identify the underlying measurements which explain the computed inequality. For example, when applying inequality indices to aggregation of SLOC values computed per class, decomposability enables one to identify the single class, or the top 10% of the classes, most responsible for the observed inequality.

However, in order to answer such questions, a MECE partitioning of the system must be given first. If one is interested, e.g., in identifying the single value most responsible for the observed inequality in the set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, there are n candidate partitionings $(\{x_i\}, \mathcal{X} \setminus \{x_i\})$,

[1]For alternative forms of decomposition, as well as, e.g., decomposition of I_{Atkinson} according to the group means rather that the equally-distributed equivalents see [16, 30]

for all $i = 1, \ldots, n$. Assuming $x_1 \leq x_i \leq x_n$, for all $i = 1, \ldots, n$, we show that it suffices to consider only the partitionings $(\{x_1\}, \mathcal{X} \setminus \{x_1\})$ and $(\{x_n\}, \mathcal{X} \setminus \{x_n\})$, since it is either the lowest (x_1) or the highest (x_n) value that contributes the most to the inequality.

We illustrate the following lemma using a definition of R specialized to the Theil index:

$$R(\{G_1, \ldots, G_m\}) = \frac{I_{\text{Theil}}^{between}(\{G_1, \ldots, G_m\})}{I_{\text{Theil}}(\mathcal{X})}, \tag{3.32}$$

where I_{Theil} is decomposed using the vector of group means (equation 3.23).

Note that R is undefined when $I_{\text{Theil}}(\mathcal{X}) = 0$, i.e., when all values being aggregated are equal. However, this is not a serious limitation since there is no inequality when all values being aggregated are equal, and explanation of inequality is only relevant in the presence of it.

Lemma 3.2.1. *Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ be a collection of values such that $x_1 \leq x_i \leq x_n$, for all $i = 1, \ldots, n$. If there exist at least two distinct values in $\mathcal{X}$ (R is defined), then it is either x_1 or x_n that contributes the most to the inequality measured using I_{Theil}, i.e., it is either the partitioning $(\{x_1\}, \mathcal{X} \setminus \{x_1\})$ or the partitioning $(\{x_n\}, \mathcal{X} \setminus \{x_n\})$ that provides the best explanation for the inequality measured using I_{Theil}.*

Proof. We distinguish between two cases: either $x_i \leq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$ or $x_i \geq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$. We show for all $i = 2, \ldots, n-1$ that if $x_i \leq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, then the partitioning $(\{x_1\}, \mathcal{X} \setminus \{x_1\})$ provides a better explanation for the inequality measured using I_{Theil} than partitioning $(\{x_i\}, \mathcal{X} \setminus \{x_i\})$, i.e., we show

$$R\left(\{x_1\}, \mathcal{X} \setminus \{x_1\}\right) \geq R\left(\{x_i\}, \mathcal{X} \setminus \{x_i\}\right), \tag{3.33}$$

for all $i = 2, \ldots, n-1$. Similarly, one can show that if $x_i \geq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, then the partitioning $(\{x_n\}, \mathcal{X} \setminus \{x_n\})$ provides a better explanation for the inequality measured using I_{Theil} than any partitioning $(\{x_i\}, \mathcal{X} \setminus \{x_i\})$, for all $i = 2, \ldots, n-1$, i.e.,

$$R\left(\{x_n\}, \mathcal{X} \setminus \{x_n\}\right) \geq R\left(\{x_i\}, \mathcal{X} \setminus \{x_i\}\right),$$

for all $i = 2, \ldots, n-1$.

By definition,

$$R\left(\{x_1\}, \mathcal{X} \setminus \{x_1\}\right) = \frac{I_{\text{Theil}}^{between}\left(\{x_1\}, \mathcal{X} \setminus \{x_1\}\right)}{I_{\text{Theil}}(\{x_1, x_2, \ldots, x_n\})},$$

and

$$R\left(\{x_i\}, \mathcal{X} \setminus \{x_i\}\right) = \frac{I_{\text{Theil}}^{between}\left(\{x_i\}, \mathcal{X} \setminus \{x_i\}\right)}{I_{\text{Theil}}(\{x_1, x_2, \ldots, x_n\})}$$

Since there exist at least two distinct values in the set $\{x_1, x_2, \ldots, x_n\}$, $I_{\text{Theil}}(\{x_1, x_2, \ldots, x_n\}) \neq 0$. It follows that proving (3.33) is equivalent to proving

$$I_{\text{Theil}}^{between}\left(\{x_1\}, \mathcal{X} \setminus \{x_1\}\right) \geq I_{\text{Theil}}^{between}\left(\{x_i\}, \mathcal{X} \setminus \{x_i\}\right)$$

Let G be the group $\mathcal{X} \setminus \{x_1\} = \{x_2, \ldots, x_n\}$, and let $mean(G)$ be the mean of the values in G, i.e., $mean(G) = \frac{\sum_{j=2}^{n} x_j}{n-1}$. Similarly, let G' be the group $\mathcal{X} \setminus \{x_i\} = \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$, and let $mean(G')$ be the mean of the values in G', i.e., $mean(G') = \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$. Since the inequality between groups is invariant to the dispersion of the distribution within each group, each value in each group can be replaced by the mean value of the group [101]. Then

$$\begin{aligned} I_{\text{Theil}}^{between}\left(\{x_1\}, \mathcal{X} \setminus \{x_1\}\right) &= I_{\text{Theil}}^{between}\left(\{x_1\}, \{x_2, \ldots, x_n\}\right) \\ &= I_{\text{Theil}}\left(mean(\{x_1\}), mean(G), \ldots, mean(G)\right) \\ &= I_{\text{Theil}}\left(x_1, \frac{\sum_{j=2}^{n} x_j}{n-1}, \ldots, \frac{\sum_{j=2}^{n} x_j}{n-1}\right) \end{aligned}$$

However, I_{Theil} satisfies the principle of transfers, i.e., a transfer of $\delta \geq 0$ from a richer to a poorer individual, without reversing their roles, does not increase inequality.

Let $\delta = \frac{x_i - x_1}{n-1}$. Since $x_1 \leq x_i \leq x_n$, for all $i = 1, \ldots, n$, then $\delta \geq 0$. We subsequently transfer δ from each $mean(G)$ to x_1. Since $x_i \leq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, then the subsequent $n-1$ transfers of size δ to x_1 do not reverse the roles between x_1 and any of the $mean(G)$. Indeed, after all the transfers x_1 becomes

$$x_1 + (n-1) \cdot \delta = x_1 + (n-1)\frac{x_i - x_1}{n-1} = x_i,$$

and each of $mean(G)$ becomes

$$\begin{aligned} mean(G) - \delta &= \frac{\sum_{j=2}^{n} x_j}{n-1} - \delta \\ &= \frac{\sum_{j=2}^{n} x_j}{n-1} - \frac{x_i - x_1}{n-1} \\ &= \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1} \end{aligned}$$

Because $x_i \leq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, then the subsequent $n-1$ transfers of δ from each $mean(G)$ to x_1 do not reverse their roles in the hierarchy, since

$$x_1 + (n-1) \cdot \delta \leq mean(G) - \delta$$

It follows that

$$
\begin{aligned}
& I_{\text{Theil}}\left(x_1, \frac{\sum_{j=2}^{n} x_j}{n-1}, \ldots, \frac{\sum_{j=2}^{n} x_j}{n-1}\right) \\
& \geq I_{\text{Theil}}\left(x_1 + \delta, \frac{\sum_{j=2}^{n} x_j}{n-1} - \delta, \ldots, \frac{\sum_{j=2}^{n} x_j}{n-1}\right) \\
& \geq \ldots \\
& \geq I_{\text{Theil}}\left(x_1 + (n-1)\cdot\delta, \frac{\sum_{j=2}^{n} x_j}{n-1} - \delta, \ldots, \frac{\sum_{j=2}^{n} x_j}{n-1} - \delta\right) \\
& = I_{\text{Theil}}\left(x_i, \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}, \ldots, \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}\right) \\
& = I_{\text{Theil}}\left(mean(\{x_i\}), mean(G'), \ldots, mean(G')\right) \\
& = I_{\text{Theil}}^{between}\left(\{x_i\}, \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}\right) \\
& = I_{\text{Theil}}^{between}\left(\{x_i\}, \mathcal{X} \setminus \{x_i\}\right)
\end{aligned}
$$

If $x_i \leq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, we have shown that the partitioning $(\{x_1\}, \mathcal{X} \setminus \{x_1\})$ provides a better explanation for the inequality measured using I_{Theil} than any partitioning $(\{x_i\}, \mathcal{X} \setminus \{x_i\})$, for all $i = 2, \ldots, n-1$. Similarly, if $x_i \geq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, then the partitioning $(\{x_n\}, \mathcal{X} \setminus \{x_n\})$ provides a better explanation for the inequality measured using I_{Theil} than any partitioning $(\{x_i\}, \mathcal{X} \setminus \{x_i\})$, for all $i = 2, \ldots, n-1$.

Since for any $x_i \in \mathcal{X}$, either $x_i \geq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$ or $x_i \leq \frac{\sum_{j=1, j\neq i}^{n} x_j}{n-1}$, we conclude that when one is interested in identifying the single value most responsible for the observed inequality in the set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, then it suffices to consider only the partitionings $(\{x_1\}, \mathcal{X} \setminus \{x_1\})$ and $(\{x_n\}, \mathcal{X} \setminus \{x_n\})$, since it is either the lowest (x_1) or the highest (x_n) value that contributes the most to the inequality.

□

Therefore, when applying inequality indices to aggregation of SLOC values measured per class, Lemma 3.2.1 allows one to answer questions such as "*Which class contributes the most to the observed inequality?*", or "*Which classes are responsible for 80% of the observed inequality?*". The former can be answered by considering only the partitionings $(\{x_1\}, \mathcal{X} \setminus \{x_1\})$ and $(\{x_n\}, \mathcal{X} \setminus \{x_n\})$, where x_1 is the smallest class and x_n is the largest class considered, and computing the R index using equation 3.32. The class that contributes the most to the observed inequality (either x_1 or x_n) is the class for which the R index is the largest. The latter can be answered by repetitively determining the class that contributes the most to the observed inequality, removing it from the data set, and repeating the process until the desired percentage of total inequality is reached.

3.3 Discussion and summary

In this chapter we have presented a number of income inequality indices, initially used to study the inequality of income or wealth distributions. We have analyzed some of their mathematical properties and we have studied their applicability to software metrics. Table 3.3 summarizes information about domain, range, invariance (w.r.t addition or multiplication), decomposability, symmetry, principle of transfers, and population principle for the inequality indices considered.

Table 3.3: Mathematical properties of the inequality indices considered.

Index	Domain	Range	Inv.	Dec.	Sym.	Tra.	Pop.
I_{Gini}	$\mathbb{R}^n_{\bar{x}\neq 0}$	$\mathbb{R}^{\dagger}$	$*$	N	Y	Y	Y
I_{Theil}	$\mathbb{R}^n_{\forall x_i \geq 0, \exists x_i > 0}$	$[0, \log n]$	$*$	Y	Y	Y	Y
I_{MLD}	$\mathbb{R}^n_{\forall x_i \geq 0, \exists x_i > 0}$	$\mathbb{R}^{\geq 0}$	$*$	Y	Y	Y	Y
$I^{\alpha}_{\text{Atkinson}}$	$\mathbb{R}^n_{\forall x_i \geq 0, \exists x_i > 0}$	$[0, 1 - \frac{1}{n}]$	$*$	Y	Y	Y	Y
I_{Hoover}	$\mathbb{R}^n_{\bar{x}\neq 0}$	$\mathbb{R}^{\dagger}$	$*$	N	Y	N	Y
I^{β}_{Kolm}	$\mathbb{R}^n$	$\mathbb{R}^{\geq 0}$	$+$	Y	Y	Y	Y

† $\left[0, 1 - \frac{1}{n}\right]$, if $\forall x_i \geq 0, \exists x_i > 0$.

The results of the examination of the appropriateness of six inequality indices (I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, I_{Atkinson}, and I_{Kolm}) for software metrics have shown that:

- I_{Theil}, I_{MLD}, and I_{Atkinson} cannot be used to aggregate metrics with negative values (e.g., the Maintainability Index);
- The six indices do not range over the same intervals. To enable direct comparison between them, they should be normalized to the same range;
- All six inequality indices are zero when all individuals have identical income, i.e., when all metrics data values are equal;
- Invariance with respect to multiplication enables that inequality be comparable across different metrics. However, scale-invariant inequality indices (I_{Gini}, I_{Theil}, I_{MLD}, and I_{Atkinson}) do not discern between all values being equal but low, and all values being equal but high;
- An important question in interpreting the value aggregated on a system level pertains to the extent to which the result can be attributed to differences between system subcomponents. The decomposability property of I_{Theil}, I_{MLD}, I_{Atkinson}, and I_{Kolm} enables such analyses.

Additionally, decomposability enables that the influence of the partitioning on the overall inequality be quantifiable;

- The decomposability property also enables root-cause analyses up to single-value level, i.e., one can identify the single value which contributes the most to the inequality being measured, by partitioning the n values into two groups, one of size 1 and the other of size $n-1$. Although there are n such partitionings for a population of size n, we have shown for I_{Theil} that such root-cause analyses can be performed efficiently, by only considering the partitionings in which the groups of size 1 contain either the lowest, or the highest observed values;

- Decomposability studies require that the partitions used be mutually exclusive and completely exhaustive (MECE). However, partitions of a software system do not necessarily satisfy MECE, e.g., the decomposition of Java systems into packages and subpackages, in which classes may exist which are not part of any of the subpackages.

From the previous analysis it follows that when using inequality indices for aggregation of software metrics, often several choices exist between using one index or another, when all the applicability criteria are met by more indices simultaneously. To simplify the choice, we recall the observation of Allison [4] by which all scale-invariant inequality measures which satisfy the principle of transfers have a simple relation to the Lorenz curve.

Indeed, given two distributions $\mathcal{X}$ and $\mathcal{Y}$, if the Lorenz curve for $\mathcal{Y}$ is never above and is somewhere below the Lorenz curve for $\mathcal{X}$, then any scale-invariant inequality index which satisfies the principle of transfers (i.e., I_{Gini}, I_{Theil}, I_{MLD}, I_{Atkinson}) will give the same rank ordering [4, 40, 71]. However, it is not uncommon in econometrics for two Lorenz curves to intersect [34]. In such cases, e.g., I_{Theil} may give one rank ordering, while I_{Atkinson} gives another. We further investigate agreement between different inequality indices empirically in Chapters 5 and 6.

Chapter 4

Threshold-based aggregation techniques

Aggregation of software metrics, regardless of whether we refer to aggregation of different metrics applied to the same artifact, or aggregation of the same metric applied to different artifacts, has been around ever since software quality assessments started being performed [19]. For example, if we refer to aggregation of the same metric applied to different artifacts, which is also the focus of this thesis, *average number of lines of code* [61] for a class (i.e., the arithmetic mean of the number of lines of code of all the nested methods) and *Weighted Methods per Class* (WMC) [22] (i.e., the sum of McCabe's cyclomatic complexities [73] of methods defined in a class) have been around since as early as 1984 and 1994, respectively.

However, aggregation of software metrics has not been recognized as a research topic on its own until very recently [99, 113–115] when, besides traditional aggregation techniques such as *sum* or *mean*, more advanced approaches to aggregation of software metrics have been proposed.

In addition to the econometric inequality indices discussed in Chapter 3, more recent work in software quality models has proposed threshold-based approaches to aggregation of software metrics. We can distinguish between threshold-based aggregation techniques using *hard thresholds* (i.e., a software artifact is given a certain rating as long as the values of its associated metrics stay within certain boundaries), such as the approach in the Software Improvement Group (SIG) quality model [5], and threshold-based aggregation techniques using *soft thresholds* (i.e., the rating computed for a software artifact does not exhibit staircasing effects as the the values of its associated metrics change slightly), such as the approach in the Squale quality model [76].

The approach to aggregation of code-level measurements to system-level *star ratings* in the SIG quality model [5] is the most recent version to date of a series of threshold-based methodologies to aggregate software metrics (e.g.,

[27,50]). The approach to aggregation of code-level measurements to system-level *global marks* in the Squale quality model [76] improves upon previous models for software product quality that use averages or weighted averages for aggregation of software metrics (e.g., [11,20,95]), by proposing a solution that addresses some of the shortcomings associated with these aggregation techniques. Certification models such as the ones proposed in [49,117] do not describe the low-level details, metrics, and aggregation techniques involved in quantifying particular quality attributes, hence go beyond the scope of this thesis.

4.1 SIG star ratings

Aggregation of code-leve measurements to system-level *star ratings* in the SIG quality model is a two-step process

In the first step, individual measurements are aggregated into *risk profiles* using thresholds on metrics [6, 50]. A risk profile represents the percentage of total source code that falls into each of four risk categories (low, moderate, high, and very high). The thresholds used to identify the boundaries between the categories are derived empirically from the measurement data of a benchmark of 100 proprietary and open-source systems written in Java and C# [6]. For example, the thresholds derived for the McCabe *cyclomatic complexity* metric are 6, 8, and 14, i.e., the methods with McCabe in $(-\infty, 6]$ are considered low risk, $(6, 8]$ moderate risk, $(8, 14]$ high risk, and $[14, \infty)$ very high risk.

In the second step, risk profiles are further aggregated into a system-level *star rating* (on a 5-point scale) using a similar, threshold-based approach. The thresholds, this time on risk profiles rather than metrics, are again derived empirically from the same benchmark data, such that each star on the 5-point scale represents equally 20% of the systems in the benchmark.

We discuss the specifics of each step below.

4.1.1 Deriving thresholds on metrics

The methodology proposed in [6] for deriving thresholds for metrics consists of six steps, as follows. The authors illustrate each step with the examples on Azureus/Vuze below.

Extraction of metrics: Metrics are extracted from a benchmark of software systems. For each *entity* (e.g., method, class) in each *system*, the value of the *metric* is recorded, as well as the entity's *weight*, i.e., its SLOC value. For example, the *MyTorrentsView.createTabs()* method of Azureus/Vuze has a McCabe metric value of 17 and a weight (SLOC) value of 119.

Calculation of weight ratio: For each *entity* (e.g., method, class) in each *system*, the *weight ratio* is computed, i.e., the ratio between the entity weight (SLOC value) and the total weight (total SLOC) of the system. For example, the *MyTorrentsView.createTabs()* method of Azureus/Vuze represents 0.036% of the entire system.

Aggregation of entities: For each metric value, the weight ratios for all entities with that metric value are summed for each system. For example, all entities with a McCabe metric value of 17 represent 1.485% of the total size of Azureus/Vuze.

Aggregation of systems: For each metric value, the previously-aggregated weight ratios are normalized by the number of systems in the benchmark (e.g., if there are 100 systems in the benchmark and all entities with a McCabe value of 17 represent 1.485% of the total size of Azureus/Vuze, then they represent $\frac{1.485}{100}$% of all code). Next, the total weight for each metric value is computed. For example, a McCabe metric value of 17 represents 0.658% of all code.

Aggregation of weight ratios: The metric values are sorted ascendingly and a density function is computed, for which the x-axis represents the benchmark-level weight ratio (0-100%), and the y-axis represents the metric value scale. The density function plots for each percentage of the weight $(1\%, 2\%, \ldots, 100\%)$ the maximal metric value. For example, for 60% of the overall code the maximal McCabe value is 2.

Derivation of thresholds: The three thresholds delimiting the four risk categories are derived by choosing 70% (i.e., low risk between 0-70%), 80% (i.e., moderate risk between 70-80%), and 90% (i.e., high risk between 80-90% and very high risk between 90-100%) of all code. For example, for the McCabe metric the thresholds are 6, 8, and 14.

4.1.2 Computing a risk profile

Computing a risk profile starts by categorizing all methods into the four risk categories, given their metric values. Next, the *weights* for each category are computed, i.e., the sum of the sizes of all the methods in the category divided by the total size of the system. The resulting risk profile is a 4-tuple in which each element represents the percentage of total source code that falls into each of the four risk categories. For example, the risk profile for ArgoUML shows that 74.2% of the code is low risk, 7.1% moderate risk, 8.8% high risk, and 9.9% very high risk [5].

4.1.3 Deriving thresholds on risk profiles

The aggregation of risk profiles into a system-level star rating is again performed using a threshold-based approach, with thresholds for risk profiles. These thresholds are derived from the same benchmark of systems using the methodology proposed in [5], which consists of the following steps. The authors illustrate each step with the examples on ArgoUML below.

Computation of risk profiles: The risk profiles are computed for all systems in the benchmark, using the approach above.

Computation of cumulative risk profiles: The cumulative risk profiles are computed for all systems in the benchmark, i.e., the relative size of each risk category plus all higher categories. Since the cumulative relative size for the low risk category is 100% by definition, one needs to compute the cumulative relative size only for the moderate, high, and very high risk categories. It follows that for the 5-point star scale used, 4 sets of thresholds are sufficient to cover it, where each set of thresholds consists of three values for the moderate, high, and very high risk categories. For example, given the previous risk profile for ArgoUML, the cumulative relative size for the very high risk category is 9.9%, for the high risk category is 9.9% + 8.8% = 18.7%, for the moderate risk category is 9.9% + 8.8% + 7.1% = 25.8%, and for the low risk category is 9.9% + 8.8% + 7.1% + 74.2% = 100% (the latter can thus be omitted).

Choice of partition: The desired distribution of the systems per rating, i.e., the number of systems in the benchmark for each rating, is chosen. For example, for a benchmark of 100 systems and a 5-point scale with uniform distribution, each rating represents equally 20% of the systems, i.e., 20 systems are rated with one star, 20 with two stars, etc.

Derivation of thresholds: The smallest possible thresholds for the three risk categories are computed using an optimization algorithm that takes as input all cumulative risk profiles and the partition, such that the desired number of systems for that rating is preserved. Discussing the optimization algorithm goes beyond the scope of this thesis. The exact threshold values for SLOC can be found in [5].

4.1.4 Computing a star rating

A system-level *star rating* for a given system is computed by determining the set of minimal thresholds, such that these all thresholds are higher or equal than the values of the cumulative risk profile of the system. For ease of presentation, we illustrate the computation of a discrete rating. A continuous scale can be obtained by linear interpolation, as described in [5].

Let $RP = (RP_M, RP_H, RP_{VH})$ be the cumulative risk profile (RP_L is omitted since $RP_L = 100\%$). Let $\{(T^i_M, T^i_H, T^i_{VH}) \mid 1 \leq i < N\}$ be the set of thresholds on risk profiles for an N-point star scale.

Then, a star rating $R \in \{1, 2, \ldots, N\}$ is computed [5] as

$$R = N - \min(i_M, i_H, i_{VH}) + 1, \tag{4.1}$$

where

$$\begin{aligned} i_M &= \min_{1 \leq i < N} (RP_M \leq T^i_M) \\ i_H &= \min_{1 \leq i < N} (RP_H \leq T^i_H) \\ i_{VH} &= \min_{1 \leq i < N} (RP_{VH} \leq T^i_{VH}) \end{aligned} \tag{4.2}$$

For example, given the cumulative risk profile for McCabe in ArgoUML of $(25.8\%, 18.7\%, 9.9\%)$ as well as the necessary thresholds, the resulting rating is three stars. Since each star on the 5-point star corresponds to 20% of the systems in the benchmark used for deriving the thresholds, and since there are two two stars above, and two stars below the rating of ArgoUML, then this implies that 40% of the systems in the benchmark scored higher, and 40% scored lower than ArgoUML.

The drawbacks of the SIG approach are twofold. First, as noted in [76], in practice such thresholds are often company-, or even development-team-dependent inside the same company. Consequently, it is not clear if the same thresholds can be applied across various companies and development teams. Moreover, the thresholds have been derived from benchmark data containing certain releases of the systems in the benchmark, and it is not clear whether the same thresholds can be applied when analyzing subsequent releases of a software system, since the threshold values do not evolve as software evolves.

Second, by translating the individual measurements to a discrete scale (with risk profiles), minor changes in quality may not be reflected in the rating, thus discouraging small, progressive improvements. For example, if the McCabe value of all methods with McCabe 13 drops to 10 in a subsequent release of the same software system, without any other changes, the improvement in quality would not be reflected in the rating.

Note that this "staircasing" problem cannot be avoided by using the continuous variant of the SIG approach, since computing the risk profiles, when the problem becomes apparent, precedes computing the star rating, when the continuous scale can be used.

4.2 Squale global marks

Squale [76] is another quality model that introduces threshold-based formulas to aggregate metric values. While Squale covers *both* aggregation of different metrics applied to the same artifact and aggregation of the same metric applied to different artifacts, in the coming discussion we focus only on the latter, which is considered as a two-phase process.

First, values of individual metrics are translated to *individual marks* in the range $[0, 3]$, such that clearly desirable values get the highest mark (3), and clearly undesirable values get the lowest mark (0). The translation function is chosen such that when a certain threshold is exceeded, then the individual mark decreases following an exponential curve. As a result, the individual mark tends quickly towards zero, stressing the presence of undesirable metric values. Moreover, in contrast to the SIG approach, the translation is continuous, ensuring that minor changes in metric values are reflected in the aggregated result (individual mark). Figure 4.2 shows an example of such a translation function, used to transform the SLOC values computed for each method to individual marks (IMs).

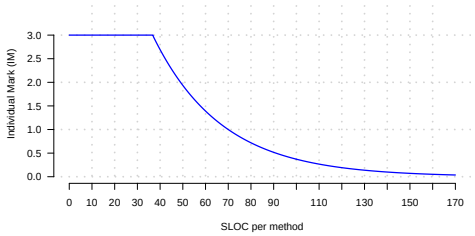

Figure 4.1: Translation function from SLOC per method to IMs.

Second, individual marks are aggregated to a system-level *global mark* (again in the range $[0, 3]$, where 3 is the perfect mark). The aggregation function (hereafter denoted as $I^{\lambda}_{\text{Squale}}$ or I_{Squale} if the value of λ is not important) is defined in terms of a parameter λ, which stresses or loosens the tolerance for bad individual marks.

$$I^{\lambda}_{\text{Squale}} = -\log_{\lambda}\left(\frac{1}{n}\sum_{i=1}^{n}\lambda^{-\text{IM}_n}\right), \tag{4.3}$$

where IM_i is the individual mark of method/class i.

Typically, three values for λ are used [76]: 30 (*hard weighing*) is associated with low tolerance for bad individual marks, i.e., the global mark falls in the range $[0, 1]$ as soon as there are a few low IMs; 9 (*medium weighing*) is associated with medium tolerance for bad individual marks, i.e., the global mark falls in the range $[0, 1]$ only when there are significantly many low IMs; 3 (*soft weighing*) is associated with large tolerance for bad individual marks, i.e., the global mark falls in the range $[0, 1]$ only when most IMs are low.

4.2.1 Mathematical properties

I_{Squale} has been introduced to aggregate non-negative values in the range $[0, 3]$ (individual marks). However, in this section we generalize it and apply I_{Squale} directly to software metrics. In this sense, we discuss mathematical properties of I_{Squale} and we compare them to the properties of the inequality indices discussed in Section 3.2.

When applied directly to software metrics, the *domain* of I_{Squale} becomes $\mathbb{R}^n$, with $\lambda \in \mathbb{R}^+ \setminus \{1\}$. Similarly, the *range* of I_{Squale} is $\mathbb{R}$. In contrast to the inequality indices that are zero when all values are equal, from the definition above (equation 4.3) it follows that $I_{\text{Squale}}(x, \ldots, x) = x$.

We start by showing that $I_{\text{Squale}}(x_1, \ldots, x_n)$ ranges always between the smallest and the largest values being aggregated. Moreover, if $\lambda > 1$, I_{Squale} is upper bound by the mean, while if $0 < \lambda < 1$ then I_{Squale} is lower bound by the mean.

Lemma 4.2.1. *Let $x_1, \ldots, x_n$ be real numbers such that $x_1 \leq x_i \leq x_n$, for all i, $1 \leq i \leq n$. Let $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Then*

$$x_1 \leq I_{\text{Squale}}(x_1, \ldots, x_n) \leq x_n,$$

for all $\lambda \in \mathbb{R}^+ \setminus \{1\}$. Moreover, if $\lambda > 1$, then

$$x_1 \leq I_{\text{Squale}}(x_1, \ldots, x_n) \leq \bar{x}$$

Similarly, if $0 < \lambda < 1$, then

$$\bar{x} \leq I_{\text{Squale}}(x_1, \ldots, x_n) \leq x_n$$

Proof. Since $x_1 \leq x_i \leq x_n$ for all $1 \leq i \leq n$, then it also holds that $-x_n \leq -x_i \leq -x_1$. Next, we distinguish between $\lambda > 1$ and $0 < \lambda < 1$.

If $\lambda > 1$, then for all i it holds that $\lambda^{-x_n} \leq \lambda^{-x_i} \leq \lambda^{-x_1}$. Therefore,

$$\begin{aligned}
& n\lambda^{-x_n} \leq \sum_{i=1}^{n} \lambda^{-x_i} \leq n\lambda^{-x_1} \\
\equiv\ & \lambda^{-x_n} \leq \frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \leq \lambda^{-x_1} \\
\equiv\ & -x_n \leq \log_\lambda \left(\frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \right) \leq -x_1
\end{aligned}$$

Hence

$$x_1 \leq I_{\text{Squale}}(x_1, \ldots, x_n) \leq x_n$$

If $0 < \lambda < 1$, then $\lambda^{-x_1} \leq \lambda^{-x_i} \leq \lambda^{-x_n}$. By the same argument,

$$\lambda^{-x_1} \leq \frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \leq \lambda^{-x_n}$$

$$\equiv \ -x_n \leq \log_\lambda \left(\frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \right) \leq -x_1$$

Hence

$$x_1 \leq I_{\text{Squale}}(x_1, \ldots, x_n) \leq x_n$$

Now, recall that the geometric mean never exceeds the arithmetic mean.

$$\sqrt[n]{\prod_{i=1}^{n} \lambda^{-x_i}} \leq \frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i}$$

However, $\sqrt[n]{\prod_{i=1}^{n} \lambda^{-x_i}} = \lambda^{-\frac{1}{n}\sum_{i=1}^{n} x_i} = \lambda^{-\bar{x}}$. Hence,

$$\lambda^{-\bar{x}} \leq \frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i}$$

If $\lambda > 1$, then

$$-\bar{x} \leq \log_\lambda \left(\frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \right)$$

and

$$-\log_\lambda \left(\frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \right) \leq \bar{x},$$

i.e.,

$$I_{\text{Squale}}(x_1, \ldots, x_n) \leq \bar{x}$$

Similarly, if $0 < \lambda < 1$, then

$$-\bar{x} \geq \log_\lambda \left(\frac{1}{n}\sum_{i=1}^{n} \lambda^{-x_i} \right)$$

and

$$\bar{x} \leq I_{\text{Squale}}(x_1, \ldots, x_n)$$

□

From the software engineering point of view, Lemma 4.2.1 has the following implications. If one is interested in highlighting the low values in the data set, i.e., low values are undesirable, then a parameter $\lambda > 1$ should be chosen. Such situations arise, e.g., when aggregating individual marks into a global mark. Similarly, if one is interested in highlighting the high values in the data set, i.e., high values are undesirable, then a parameter $0 < \lambda < 1$ should be chosen. Such situations arise, e.g., when applying I_{Squale} directly to aggregation of SLOC values. Moreover, the Squale aggregation is more sensitive to undesirable values than the mean.

The next lemma relates the aggregated values for different values of λ.

Lemma 4.2.2. $I^{\lambda^k}_{\text{Squale}}(x_1, \ldots, x_n) = \frac{1}{k} I^{\lambda}_{\text{Squale}}(kx_1, \ldots, kx_n)$, *for all* $k \in \mathbb{R} \setminus \{0\}$ *and* $\lambda \in \mathbb{R}^+ \setminus \{1\}$.

Proof.

$$\begin{aligned} I^{\lambda^k}_{\text{Squale}}(x_1, \ldots, x_k) &= -\log_{\lambda^k}\left(\frac{1}{n}\sum_{i=1}^{n}(\lambda^k)^{-x_i}\right) \\ &= -\frac{1}{\log_\lambda \lambda^k}\log_\lambda\left(\frac{1}{n}\sum_{i=1}^{n}(\lambda^k)^{-x_i}\right) \\ &= \frac{1}{k}\left(-\log_\lambda\left(\frac{1}{n}\sum_{i=1}^{n}\lambda^{k(-x_i)}\right)\right) \\ &= \frac{1}{k} I^{\lambda}_{\text{Squale}}(kx_1, \ldots, kx_n) \end{aligned}$$

□

It follows that one can choose $\lambda = e$ as the standard instantiation of the Squale family of indices.

Lemma 4.2.3. $I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_n) = \frac{1}{\log \lambda} I^{e}_{\text{Squale}}(x_1 \log \lambda, \ldots, x_n \log \lambda)$

Proof. From Lemma 4.2.2 we have

$$I^{\lambda^k}_{\text{Squale}}(x_1, \ldots, x_n) = \frac{1}{k} I^{\lambda}_{\text{Squale}}(kx_1, \ldots, kx_n),$$

for all $k \in \mathbb{R} \setminus \{0\}$ and $\lambda \in \mathbb{R}^+ \setminus \{1\}$. Let $\lambda = e$. Then

$$I^{e^k}_{\text{Squale}}(x_1, \ldots, x_n) = \frac{1}{k} I^{e}_{\text{Squale}}(kx_1, \ldots, kx_n)$$

Let $k = \log \lambda$. Then

$$I^{e^{\log \lambda}}_{\text{Squale}}(x_1, \ldots, x_n) = \frac{1}{\log \lambda} I^{e}_{\text{Squale}}(x_1 \log \lambda, \ldots, x_n \log \lambda)$$

Hence

$$I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_n) = \frac{1}{\log \lambda} I^{e}_{\text{Squale}}(x_1 \log \lambda, \ldots, x_n \log \lambda)$$

□

In addition, similar to the inequality indices, I_{Squale} is *symmetric* and satisfies the *population principle*. The first statement follows immediately from the definition of I_{Squale}. We prove the second statement next.

Lemma 4.2.4. *Let $y_1, \ldots, y_m$ be a replication of $x_1, \ldots, x_n$, i.e., let there exist k such that $m = nk$ and $y_j = x_{\lceil j/k \rceil}$ for all j, $1 \leq j \leq m$. Then, I_{Squale} satisfies the population principle, i.e.,*

$$I^{\lambda}_{\text{Squale}}(y_1, \ldots, y_m) = I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_n)$$

Proof.

$$\begin{aligned}
I^{\lambda}_{\text{Squale}}(y_1, \ldots, y_m) &= -\log_{\lambda} \left(\frac{1}{m} \sum_{j=1}^{m} \lambda^{-y_j} \right) \\
&= -\log_{\lambda} \left(\frac{1}{nk} \sum_{j=1}^{nk} \lambda^{-y_j} \right) \\
&= -\log_{\lambda} \left(\frac{1}{nk} \sum_{j=1}^{nk} \lambda^{-x_{\lceil j/k \rceil}} \right) \\
&= -\log_{\lambda} \left(\frac{1}{nk} \sum_{i=1}^{n} k \lambda^{-x_i} \right) \\
&= -\log_{\lambda} \left(\frac{k}{nk} \sum_{i=1}^{n} \lambda^{-x_i} \right) \\
&= -\log_{\lambda} \left(\frac{1}{n} \sum_{i=1}^{n} \lambda^{-x_i} \right) \\
&= I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_n)
\end{aligned}$$

□

Next we compare I_{Squale} with I_{Kolm} [60], introduced in Chapter 3. Recall the definition of I^{β}_{Kolm}:

$$I^{\beta}_{\text{Kolm}}(x_1, \ldots, x_n) = \frac{1}{\beta} \log \left(\frac{1}{n} \sum_{i=1}^{n} e^{\beta(\bar{x} - x_i)} \right)$$

where $\bar{x}$ is the mean of $x_1, \ldots, x_n$.

The following lemma establishes a formal relation between I_{Squale} and I_{Kolm}.

Lemma 4.2.5. $I^{\log\lambda}_{\text{Kolm}}(x_1,\ldots,x_n) + I^{\lambda}_{\text{Squale}}(x_1,\ldots,x_n) = \bar{x}$

Proof. To see that the lemma holds, observe that

$$\begin{aligned}
&I^{\log\lambda}_{\text{Kolm}}(x_1,\ldots,x_n) + I^{\lambda}_{\text{Squale}}(x_1,\ldots,x_n) \\
&\qquad = \frac{1}{\log\lambda}\log\left(\frac{1}{n}\sum_{i=1}^{n} e^{\log\lambda(\bar{x}-x_i)}\right) - \log_\lambda\left(\frac{1}{n}\sum_{i=1}^{n}\lambda^{-x_i}\right) \\
&\qquad = \frac{1}{\log\lambda}\log\left(\frac{1}{n}\sum_{i=1}^{n}\lambda^{\bar{x}-x_i}\right) - \frac{1}{\log\lambda}\log\left(\frac{1}{n}\sum_{i=1}^{n}\lambda^{-x_i}\right) \\
&\qquad = \frac{1}{\log\lambda}\left(\log\left(\frac{1}{n}\lambda^{\bar{x}}\sum_{i=1}^{n}\lambda^{-x_i}\right) - \log\left(\frac{1}{n}\sum_{i=1}^{n}\lambda^{-x_i}\right)\right) \\
&\qquad = \frac{1}{\log\lambda}\left(\log\frac{\frac{1}{n}\lambda^{\bar{x}}\sum_{i=1}^{n}\lambda^{-x_i}}{\frac{1}{n}\sum_{i=1}^{n}\lambda^{-x_i}}\right) \\
&\qquad = \frac{1}{\log\lambda}\left(\log\lambda^{\bar{x}}\right) \\
&\qquad = \bar{x}
\end{aligned}$$

□

Using Lemma 4.2.5 we can establish a number of additional properties of I_{Squale} for any $x_1,\ldots,x_n \in \mathbb{R}^n$. Without loss of generality, let $x_1 \le x_i \le x_n$, for all $1 \le i \le n$.

Lemma 4.2.6. *For all $c \in \mathbb{R}$ it holds that $I^{\lambda}_{\text{Squale}}$ is "unit translatable", i.e.,*

$$I^{\lambda}_{\text{Squale}}(x_1+c,\ldots,x_n+c) = I^{\lambda}_{\text{Squale}}(x_1,\ldots,x_n) + c$$

Proof. From Lemma 4.2.5 we have

$$I^{\lambda}_{\text{Squale}}(x_1+c,\ldots,x_n+c) = mean(x_1+c,\ldots,x_n+c) - I^{\log\lambda}_{\text{Kolm}}(x_1+c,\ldots,x_n+c)$$

But

$$mean(x_1+c,\ldots,x_n+c) = mean(x_1,\ldots,x_n) + c$$

Since I^{β}_{Kolm} is invariant with respect to addition [60], then

$$I^{\log\lambda}_{\text{Kolm}}(x_1+c,\ldots,x_n+c) = I^{\log\lambda}_{\text{Kolm}}(x_1,\ldots,x_n)$$

It follows that

$$\begin{aligned}
I^{\lambda}_{\text{Squale}}(x_1+c,\ldots,x_n+c) &= mean(x_1,\ldots,x_n) + c - I^{\log\lambda}_{\text{Kolm}}(x_1,\ldots,x_n) \\
&= I^{\lambda}_{\text{Squale}}(x_1,\ldots,x_n) + c
\end{aligned}$$

□

Lemma 4.2.7. *Let $x_i < x_j$ and let $\delta > 0$ be such that $x_i + \delta \leq x_j - \delta$. Then, $I^{\lambda}_{\text{Squale}}$ satisfies the "anti-transfers principle", i.e.,*

$$I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) < I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_i + \delta, \ldots, x_j - \delta, \ldots, x_n)$$

Proof. Since I_{Kolm} is known to satisfy the transfers principle [60], then for any β it holds that

$$I^{\beta}_{\text{Kolm}}(x_1, \ldots, x_i, \ldots, x_j, x_n) > I^{\beta}_{\text{Kolm}}(x_1, \ldots, x_i + \delta, \ldots, x_j - \delta, \ldots, x_n),$$

for x_i, x_j and δ as above.

From Lemma 4.2.5 we have

$$I^{\log \lambda}_{\text{Kolm}}(x_1, \ldots, x_n) = mean(x_1, \ldots, x_n) - I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_n),$$

and

$$\begin{aligned} I^{\log \lambda}_{\text{Kolm}}(x_1, \ldots, & x_i + \delta, \ldots, x_j - \delta, \ldots, x_n) \\ &= mean(x_1, \ldots, x_i + \delta, \ldots, x_j - \delta, \ldots, x_n) \\ &\quad - I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_i + \delta, \ldots, x_j - \delta, \ldots, x_n) \\ &= mean(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) \\ &\quad - I^{\lambda}_{\text{Squale}}(x_1, \ldots, x_i + \delta, \ldots, x_j - \delta, \ldots, x_n) \end{aligned}$$

The claim follows. □

From the software engineering point of view, unit translatability allows one to calculate I_{Squale} if, e.g., headers containing the licensing information have been added to all source files. In this case, the value of I_{Squale} applied to lines of code per file for the modified system equals the value of I_{Squale} applied to lines of code per file for the system before the modification, increased by the number of lines in the header.

The "anti-transfers principle" is symmetric to the transfers principle of I_{Kolm}. Intuitively, the anti-transfers principle means that I_{Squale} measures *equality* as opposed to I_{Kolm} (or any of the other inequality indices satisfying the principle of transfers, i.e., I_{Gini}, I_{Theil}, I_{MLD}, and I_{Atkinson}) measuring inequality. In this sense, transferring money from a reacher person to a poorer person increases I_{Squale}, i.e., increases equality, whereas the same transfer decreases I_{Kolm}, i.e., decreases inequality.

In software engineering, if source lines of code values measured per method are considered undesirable when greater than 36 [76], then a decrease in SLOC of 20 for one method with SLOC 60 at the cost of an equivalent increase in SLOC for another method with SLOC 20 would result in an increase of quality as measured by I_{Squale}, i.e., a decrease in the number of undesirable values.

Equally-distributed equivalent

The intuition that I_{Squale} measures equality can be further formalized by considering the econometric notion of an equally-distributed equivalent, discussed in Section 3.2.

Recall that given a social-evaluation function $W : \mathbb{R}^n \to \mathbb{R}$ and the distribution of income $x_1, \ldots, x_n$, the equally-distributed-equivalent ξ is determined by $W(x_1, \ldots, x_n) = W(\xi, \ldots, \xi)$.

Blackorby et al. [16] have shown that for an important class of social-evaluation functions[1], the corresponding equally-distributed equivalent can be written as

$$\xi^g(x_1, \ldots, x_n) = g^{-1}(\frac{1}{n}\sum_{i=1}^{n} g(x_i)), \tag{4.4}$$

for a function g.

Clearly, $I^{\lambda}_{\text{Squale}}$ belongs to this family of equally-distributed equivalents for $g(x) = \lambda^{-x}$.

4.3 Discussion and summary

In this chapter we have presented and analyzed two threshold-based approaches to aggregation of software metrics. The results of our analysis have shown that:

- Aggregation of metrics values using a discrete threshold-based approach (e.g., the aggregation of individual measurements to risk profiles in the SIG approach) can lead to small, progressive improvements in quality not being reflected in the rating, and should thus be avoided. In contrast, the Squale approach provides a continuous model for metric aggregation.

- Thresholds should be derived and validated for both the SIG approach, as well as the Squale approach to be correctly applicable. Moreover, it may be the case that thresholds should co-evolve as systems evolve.

- A high rating obtained with either the SIG or the Squale approaches is not necessarily an indication of good software engineering practices. For example, [51] recommends seven as the guideline for the maximal *depth of inheritance tree* (DIT) [22]. However, even though when all *depth of inheritance tree* (DIT) values are 0 or 1 the recommendation

[1] More precisely, the social-evaluation function W should be continuous, S-concave, increasing, the population should have not less than 3 individuals, and that some non-singleton strict subset of the population is separable from its complement. Discussion of these notions in further detail goes beyond the scope of this thesis.

of [51] is followed, such a situation suggests that the designers might not be taking advantage of reuse of methods through inheritance [22]. Hence, DIT uncovers an essential shortcoming of the threshold-based methods: the threshold-based methods assume that if all values are "good" (i.e., low risk, desirable), then also the entire system is "good", while this assumption does not hold for DIT.

- Although designed for software metrics, the aggregation of individual marks to a global mark in the Squale approach (I_{Squale}) satisfies a number of common and complementary properties of the econometric inequality indices, e.g., the *population principle* and the *anti-transfers principle*. Moreover, I_{Squale} can be formalized using the econometrical notion of *equally-distributed equivalent*, and has a formal relation with I_{Kolm}.

Concluding remarks

In the previous three chapters we have performed a theoretical analysis of three categories of aggregation techniques for software metrics, and we have discussed their applicability to the task in hand. In this section we provide an overview of our findings.

The first category of aggregation techniques for software metrics considered comprises standard summary statistics such as the *sum*, *mean*, or *median*, commonly used to aggregate software metrics [12,20,63,64,87]. The *mean* meaningfully captures the central tendency in a data set as long as the distribution is close to normal. However, it is common for software metrics to have strongly-skewed distributions [14, 46, 113], for which the *mean* fails to accurately capture the central tendency, thus proving inappropriate for aggregation of such metrics data.

To illustrate the problems with using the *median* as aggregation technique for software metrics we refer to the example in [113]. Before a refactoring of Lucene, a text search engine library written in Java, the *median* of the strongly-skewed distribution of *cyclomatic complexity* at class level was five. After the refactoring, new classes were added to the system and the *median* increased to eight, wrongly suggesting that overall *cyclomatic complexity* in Lucene increased, when in fact the increase in *median* was only due to the increase in population size.

Such strongly-skewed distributions are more accurately tackled with measures of asymmetry and peakedness, e.g., *skewness* and *kurtosis*. However, as opposed to statistics such as *sum* or *mean* which have been known to associate with software quality attributes (e.g., *mean* applied to the *number of lines of code added per revision* is associated with changeability [77]), *skewness* and *kurtosis* are used only to characterize distributions of software metrics [12, 53, 58, 100], and lack association with any quality attribute.

The second category of aggregation techniques for software metrics considered consists of econometric inequality indices, commonly used to study inequality of income or welfare distributions. The inequality indices have been designed for strongly-skewed distributions such as the distributions of income or wealth, and are thus appropriate for application to strongly-skewed software metrics.

Even though they also lack established association with quality attributes, inequality indices are known to provide insights into software quality issues. For example, the typical range for I_{Gini} applied to various class-level metrics was reported to be between 0.45 and 0.75, while values greater than 0.85 were a strong indicator for the presence of machine-generated code [113]. Similar associations between high values for I_{Theil} and presence of machine-generated code have been reported in [99]. However, since I_{Theil} is unbounded, analyses of its typical range require its normalization first.

Nonetheless, despite that high values for inequality indices applied to various metrics have been known to provide insights into software quality issues, we stress that measuring inequality in the values of quality metrics is conceptually different than measuring quality. Therefore, interpretation of the results of applying inequality indices to software metrics rests on the expertise of the analyst and in the particular characteristics of the project under examination rather than the statistic itself.

A serious drawback to inequality indices becomes apparent when applied to equal values (or values within a small range, i.e., when the ratio between the maximum and the minimum values in the data set is close to one). In such cases, regardless of whether the values are all high or all low, all inequality indices will be zero (or have very small values). However, decomposable inequality indices offer two important benefits. On the one hand, such indices provide means to *explain* the inequality of the metric values by their partitioning into mutually-exclusive and completely-exhaustive groups. On the other hand, decomposable inequality indices offer traceability and provide means to perform root-cause analyses at the level of individual values.

The quality of software is better captured by the third category of aggregation techniques for software metrics considered, i.e., the threshold-based approaches. Nonetheless, even though such techniques result in a system-level quality evaluation result, high ratings may not necessarily also indicate good software engineering practices, as it is, e.g., the case with DIT.

On the other hand, such techniques can be adapted to the requirements of each company performing the quality assessments (e.g., Squale), and provide means to trace the results back to individual measurements (e.g., SIG). However, the discrete mapping of metric results in the SIG approach introduces staircase and threshold effects, that may hide detailed information and trigger wrong interpretation. In this sense, small, progressive improvements in the values of the metrics may not be reflected in the result if they remain in the same intervals. Moreover, even though there are generally-accepted guidelines for the maximal values of metrics such as DIT (e.g., in [51]) or *cyclomatic complexity* (e.g., in [73]), it is not a priori clear that for metrics such as SLOC thresholds for the maximally-acceptable values remain constant as systems evolve. Therefore, when applying either the SIG or the Squale threshold-based approaches, one should also consider the validity of thresholds.

Part II

Empirical analysis

Introduction

The second part of the thesis discusses the results of various empirical analyses of the aggregation techniques in part one. The two studies [114, 115] which form the basis for Chapter 5 represent pilot empirical comparisons of different aggregation techniques for software metrics. In [114] we aggregate SLOC values from class to package level, and study statistical correlation between the aggregated values and the *number of defects per package*, as well as statistical correlation between the various aggregation techniques, on a single snapshot of ArgoUML. Later, in [115], we investigate three case studies (ArgoUML, Adempiere, and Mogwai) by means of an augmented list of aggregation techniques, and we focus on studying statistical correlation between the aggregated values and the *number of defects per package*.

The pilot studies are conducted in order to assess the feasibility of performing similar large-scale studies, and to distil requirements for the tooling to facilitate them. As a result, we propose tooling satisfying these requirements (Appendix A), and we conclude that future studies with the number of defects as validation metric are infeasible unless data quality can be ensured. Therefore, in the remaining part of the empirical analysis we concentrate on statistical correlation between the various aggregation techniques, at different aggregation levels, and using different metrics.

Specifically, we present the results of an extensive correlation study based on [116] of the traditional and econometric aggregation techniques in Chapter 6, in which we aggregate size (SLOC, LOC, NOS, NOSt), low-variance (DIT, NOC), and limited-range (PBS, PLwC) metrics from class to package level on the 106 systems comprising the Qualitas Corpus [110].

Later, in Chapter 7 we change the aggregation level from class–package to method–class in order to enable the comparison with the threshold-based aggregation techniques, and we present the results of a correlation study of all three categories of aggregation techniques discussed in the first part. For the purpose of illustration, we choose SLOC as metric.

Finally, aggregation techniques which are part of software quality models (e.g., Squale) are usually designed to highlight undesirable metrics values in order to warn the software engineers in case of potential problems. In Chapter 8 we empirically compare inequality indices to the aggregation technique in Squale in their ability to highlight undesirable values in the aggregate.

Chapter 5

Pilot studies

5.1 Introduction

Fault prediction models usually employ software metrics which were previously shown to be a strong predictor for defects [18, 45, 77, 79, 81]. Such a metric is size, e.g., measured in *source lines of code* (SLOC) [38]. Size (SLOC) not only corresponds to the intuitive belief that large systems have more faults in them than small systems, but was shown in [38] to act as an early indicator of problems better than, e.g., object-oriented metrics such as the Chidamber and Kemerer suite [22] or the Lorenz and Kidd suite [65]. Indeed, although such metrics had been validated with respect to fault-proneness of classes (defects), studies prior to [38] did not check for the confounding effect of size. After controlling for size, none of the above metrics could be associated with defects anymore. Hence SLOC remains a reliable and easily-measurable predictor for defects.

In this study we aggregate SLOC values from class to package level using the traditional aggregation techniques and the inequality indices introduced in Chapters 2 and 3, respectively. Specifically, we consider the *mean*, *median*, *sum*, *standard deviation*, *variance*, *skewness*, and *kurtosis*, as well as I_{Gini}, I_{Hoover}, I_{Theil}, I_{MLD}, I_{Atkinson}, and I_{Kolm}. We exclude the threshold-based aggregation techniques in Chapter 4 from the pilot study since neither the SIG, nor the Squale approaches have published thresholds for the *number of source lines of code* measured per class. Additionally, although the Squale approach could be in principle applied directly to metrics data, preliminary experiments suggested that it is unfeasible to apply Squale to aggregation of large metrics values since the larger the value, the smaller the influence it has on the aggregated results (I_{Squale} is nominally applied to individual marks, i.e., normalized metrics values in the range $[0, 3]$).

Based on the assumption that size is a good predictor for defects, hence size and defects should be statistically correlated, we wish to understand (i) whether the aggregation technique influences the presence and strength

of this relation, and (ii) whether and which aggregation techniques convey the same information and are thus redundant.

As case studies we have chosen ArgoUML[1], a popular UML modeling tool, Adempiere[2], an open-source ERP application, and Mogwai Java Tools[3], a Java Entity Relationship design and modeling (ERD) application. All three are written in Java. ArgoUML is one of the most studied open-source software systems ever, e.g., with four references at the 2011 European Conference on Software Maintenance and Reengineering only [15, 26, 44, 103]. Adempiere has been studied before on several occasions [41,99]. Both Adempiere and Mogwai Java Tools have been identified as amenable to such studies [48] since they satisfy a number of desirable criteria (e.g., have a sufficient number of developers or a sufficiently advanced development status, and accurately keep track of bugs).

5.2 Methodology

In order to answer the two questions in Section 5.1, we aggregated SLOC values from class to package level, and we conducted two series of experiments. In the first set of experiments we studied statistical correlation between the aggregated metrics and the defects and we answered question (i). In the second set of experiments we studied statistical correlation between the aggregated metrics themselves and we answered question (ii).

Both sets of experiments adhere to the methodology described below. Although we illustrate the data collection process with examples from ArgoUML, we summarize the characteristics of all three cases in Table 5.1

5.2.1 Data collection

To perform the two sets of experiments we started by choosing the case (ArgoUML, Adempiere, Mogwai) *version* with the highest number of bug fixes. The choice for bug *fixes* rather than *reports*, *dismissals* etc. is motivated by the fact that commit messages only contain (at best) information about the fixed bugs (typically indicated by keywords such as "issue" or "fix" followed by the ID of the issue in the issue tracker). This information is needed in order to associate bugs with Java classes. Moreover, this follows the approach described in [37]. Since we only analyze a snapshot of each case, the choice for the faultiest version ensures that the defect population is sufficiently large for the analysis to be accurate. For example, from the approximately 150 versions of ArgoUML released throughout its history, the version 0.13.4 has the highest number of bug fixes associated with it (89). It contains 94 packages and 1230 classes.

[1] http://http://argouml.tigris.org/

[2] http://www.adempiere.com/index.php/ADempiere_ERP

[3] http://sourceforge.net/projects/mogwai/

Next, the source code of the selected versions was automatically processed and the the list of packages and Java classes contained in each package was extracted. For each package in each of the three systems, we aggregate the SLOC values of all the classes *directly* contained in that package, in turn, using each of the aggregation techniques considered. We say that a class C is directly contained in a package P if there exists no subpackage P' of P different from P such that C is contained in P'.

In all experiments we considered packages containing at least two classes. This is motivated by the fact that when applied to packages containing only one class, all inequality indices are equal to zero. For example, 77 packages meet this requirement for version 0.13.4 of ArgoUML, from a total of 94.

In the following step we manually mapped the defects (bug fixes) to Java packages by analyzing the commit messages of the version control system log. Since the same class could have been affected multiple times during the fix of a known bug (e.g., because of a wrongly-implemented fix the first time), we only recorded it once in order to minimize noise. For example, out of the 89 issues associated with version 0.13.4 of ArgoUML, there are only 42 issues mentioned in the commit log (e.g., because some of the issues required changes to non-Java source files). The cardinality of the defect sets per package generated a list containing an element for each of the packages, and served as our validation metric.

5.2.2 Data analysis

To measure statistical correlation, either between aggregation techniques and defects, or between values aggregated using different techniques, we have a choice between linear or rank correlation coefficients.

Linear coefficients (e.g., Pearson [86]) are sensitive only to a linear relation between two variables. On the other hand, rank coefficients (e.g., Kendall [57] or Spearman [104]) are more robust to nonlinear relations since they only measure the extent to which an increase in one variable (not necessarily linear) corresponds to an increase in the other variable. To illustrate the difference between linear and rank correlations, consider the four data

Table 5.1: Summary of the characteristics of the three cases.

	ArgoUML	Adempiere	Mogwai
Version considered	0.13.4	4.1.0	2.6.0
Number of Java classes	1230	4237	2310
Number of packages	94	161	365
Number of dense packages (≥ 2 classes)	77	133	271
Number of bugs considered	89	303	143
Number of bugs mapped	42	213	38

sets of [7] in Figure 5.1, which have the same *mean*, *variance*, Pearson correlation coefficient, and regression line, yet are substantially different.

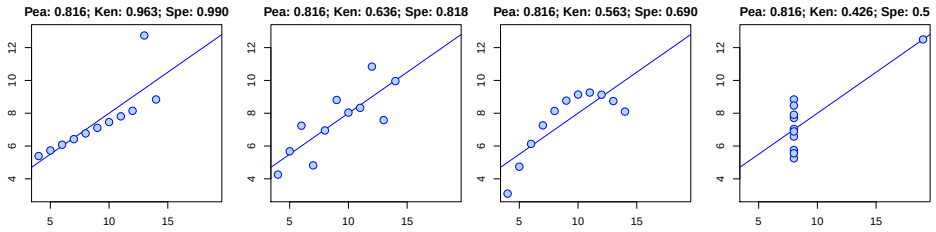

Figure 5.1: Data sets of [7]: Pearson's r is constant, while Kendall's τ and Spearman's ρ decrease and reflect the nonlinearity of the relation.

Consequently, we use a rank correlation coefficient and we opt for Kendall's τ since Spearman's ρ is known to be difficult to interpret [80]. We account for ties as described in [91]. Using Kendall's τ, in the first set of experiments we measure correlation in turn, between SLOC data aggregated from class to package level using the different aggregation techniques, and defects. Similarly, in the second set of experiments we measure correlation between SLOC data aggregated from class to package level using all pairs of aggregation techniques. We stress that statistical correlations are not transitive [62], i.e., we have to consider *all pairs* of aggregation techniques[4].

All computations were performed using R [92].

5.3 Results

We discuss the results from each set of experiments separately.

5.3.1 Correlation with bugs

For correlation between SLOC and defects, the results are summarized in Table 5.2, where boldface corresponds to two-sided p-values not exceeding 0.01, and italics corresponds to those between 0.01 and 0.05.

The following observations can be derived:

- Correlation with the number of defects always ranges from very low ($\tau \simeq 0.02$ for *mean* in ArgoUML) to medium ($\tau \simeq 0.49$ for *sum* in Adempiere). None of the techniques indicates strong and also statistically significant correlation with the number of defects.
- Values aggregated using the *mean* indicate very inconsistent results. In ArgoUML *mean* shows very low correlation with defects, while in Mogwai *mean* together with *standard deviation* (0.197), *variance* (0.197),

[4]While [62] considered Pearson's correlation coefficient, their counterexample also shows lack of transitivity for Spearman's ρ and Kendall's τ.

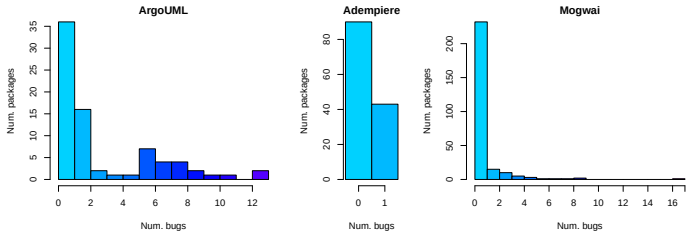

Figure 5.2: Distributions of the number of bugs per package.

and I_{Kolm} (0.204) indicate the strongest (among the techniques considered) statistically significant correlation with the number of defects.

- Values aggregated using the *sum* indicate the strongest (for ArgoUML and Adempiere) and third strongest (for Mogwai) statistically significant correlation with the number of defects. Although the correlation is not high, this confirms the intuition that large packages have more faults than small packages. Values aggregated using *skewness* indicate stronger correlation with the number of defects for ArgoUML that *sum*, but the correlation is less statistically significant.
- Values aggregated using I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson} indicate consistently similar and statistically significant correlation with the number of defects for ArgoUML and Mogwai, although none of them ever indicates the strongest correlation. ArgoUML and Mogwai are also the two systems for which the number of bugs mapped to packages is much lower that the number of packages considered (42/77 in ArgoUML, and 38/271 in Mogwai). There is also high inequality in the number of bugs per package, caused by most packages having no bugs (Figure 5.2).
- Values aggregated using *skewness* and *kurtosis* are also inconsistent, indicating very high (among the techniques considered) correlation with the number of defects for ArgoUML, and very low correlation with the number of defects for Mogwai, although ArgoUML and Mogwai are similar in the number of defects per package, i.e., they both have significantly less bugs mapped to packages than actual packages. The correlation is not statistically significant for any of the three systems.
- In the case of Adempiere there are more bugs than packages (213/133), and there is also a more egalitarian distribution of bugs per package (Figure 5.2). The inequality indices start to diverge, and the correlation with defects is less statistically significant.

To summarize our answer to question (i), correlation between SLOC and defects is not strong, and is influenced by the aggregation technique.

Table 5.2: Correlation between aggregation techniques and defects

	ArgoUML	Adempiere	Mogwai
mean	0.023	**0.411**	**0.197**
median	-0.142	**0.336**	**0.129**
sum	**0.312**	**0.490**	**0.151**
sd	*0.191*	**0.370**	**0.197**
var	*0.191*	**0.370**	**0.197**
skewness	*0.322*	0.151	0.032
kurtosis	0.291	0.104	0.099
I_{Gini}	**0.266**	**0.212**	**0.134**
I_{Theil}	**0.269**	*0.168*	**0.135**
I_{MLD}	**0.227**	*0.148*	**0.139**
I_{Hoover}	**0.239**	0.098	*0.122*
I_{Atkinson}	**0.244**	*0.149*	**0.138**
I_{Kolm}	0.144	**0.422**	**0.204**

5.3.2 Correlation between different techniques

The results of the second set of experiments are summarized in Table 5.3 for ArgoUML, Table 5.4 for Adempiere, and Table 5.5 for Mogwai, where Kendall correlation results with two-sided p-values not exceeding 0.01 are typeset in boldface, and those between 0.01 and 0.05 are typeset in italics.

Experiments on all three systems suggest that:

- One groups of aggregation techniques, consisting of I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}, is clearly distinguishable since there is high and statistically significant correlation between the aggregation techniques in the group. Across the three systems, the correlation coefficient between aggregation techniques in this group ranges between 0.81 and 0.95, and is always statistically significant.

- Another group of aggregation techniques, comprising the *mean*, *standard deviation*, and *variance* shows medium-high and statistically significant correlation with I_{Kolm} for all three systems, ranging between 0.64 and 0.85.

- The *mean* shows medium-high and statistically significant correlation with the *median*, *standard deviation*, and *variance* for all three systems, ranging between 0.57 and 0.73. However, the *mean-median* and the *mean-*I_{Kolm} pairs are a very good example of the non-transitivity of statistical correlations [62], since the correlation between the *median* and I_{Kolm} does not exceed 0.57, while the correlation between *mean* and I_{Kolm} does not drop below 0.64 for any of the systems.

- The perfect correlation coefficient between *standard deviation* and *variance* can be explained by the fact that the *standard deviation* is a monotonic transformation of the *variance*.

To summarize our answer to question (ii), our results indicate that the aggregation values obtained using I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson} convey the same information. A similar claim, although less strong, can be made for I_{Kolm}, *mean*, *standard deviation*, and *variance*.

Table 5.3: Kendall correlation results between techniques, for ArgoUML.

	med	*sum*	*sd*	*var*	*ske*	*kur*	I_{Gin}	I_{The}	I_{MLD}	I_{Hoo}	I_{Atk}	I_{Kol}
mea	**0.59**	**0.35**	**0.58**	**0.57**	0.00	-0.05	**0.22**	**0.23**	**0.31**	**0.25**	**0.27**	**0.68**
med		0.14	*0.19*	*0.19*	*-0.24*	-0.21	-0.14	-0.13	-0.044	-0.12	-0.09	**0.32**
sum			**0.41**	**0.41**	*0.26*	0.21	**0.37**	**0.36**	**0.37**	**0.32**	**0.36**	**0.46**
sd				**1.0**	*0.27*	0.14	**0.62**	**0.64**	**0.69**	**0.67**	**0.68**	**0.80**
var					*0.27*	0.14	**0.62**	**0.64**	**0.69**	**0.67**	**0.68**	**0.80**
ske						**0.82**	**0.44**	**0.48**	**0.36**	**0.43**	**0.42**	0.09
kur							0.28	*0.33*	0.22	0.26	0.28	0.00
I_{Gin}								**0.93**	**0.88**	**0.86**	**0.91**	**0.50**
I_{The}									**0.88**	**0.88**	**0.94**	**0.51**
I_{MLD}										**0.85**	**0.93**	**0.60**
I_{Hoo}											**0.89**	**0.53**
I_{Atk}												**0.56**

Table 5.4: Kendall correlation results between techniques, for Adempiere.

	med	*sum*	*sd*	*var*	*ske*	*kur*	I_{Gin}	I_{The}	I_{MLD}	I_{Hoo}	I_{Atk}	I_{Kol}
mea	**0.71**	**0.49**	**0.72**	**0.72**	0.10	0.02	**0.22**	**0.20**	**0.22**	**0.18**	**0.20**	**0.80**
med		**0.31**	**0.47**	**0.47**	-0.12	-0.13	-0.01	-0.02	0.01	-0.04	-0.01	**0.57**
sum			**0.51**	**0.51**	**0.34**	*0.27*	**0.37**	**0.33**	**0.29**	**0.24**	**0.31**	**0.54**
sd				**1.0**	**0.29**	0.21	**0.48**	**0.47**	**0.46**	**0.44**	**0.46**	**0.82**
var					**0.29**	0.21	**0.48**	**0.47**	**0.46**	**0.44**	**0.46**	**0.82**
ske						**0.84**	**0.41**	**0.42**	**0.27**	**0.36**	**0.35**	0.13
kur							*0.28*	*0.33*	0.18	0.22	0.25	0.05
I_{Gin}								**0.89**	**0.81**	**0.82**	**0.87**	**0.39**
I_{The}									**0.85**	**0.85**	**0.93**	**0.37**
I_{MLD}										**0.82**	**0.92**	**0.39**
I_{Hoo}											**0.87**	**0.34**
I_{Atk}												**0.38**

Table 5.5: Kendall correlation results between techniques, for Mogwai.

	med	*sum*	*sd*	*var*	*ske*	*kur*	I_{Gin}	I_{The}	I_{MLD}	I_{Hoo}	I_{Atk}	I_{Kol}
mea	**0.73**	**0.43**	**0.60**	**0.60**	0.01	-0.00	**0.13**	**0.14**	**0.15**	**0.16**	**0.15**	**0.64**
med		**0.28**	**0.37**	**0.37**	**-0.23**	-0.15	*-0.08*	-0.07	-0.04	-0.06	-0.05	**0.43**
sum			**0.50**	**0.50**	**0.19**	0.15	**0.41**	**0.38**	**0.37**	**0.35**	**0.38**	**0.54**
sd				**1.0**	**0.23**	0.15	**0.52**	**0.53**	**0.53**	**0.55**	**0.53**	**0.85**
var					**0.23**	0.15	**0.52**	**0.53**	**0.53**	**0.55**	**0.53**	**0.85**
ske						**0.68**	**0.41**	**0.39**	**0.29**	**0.38**	**0.34**	0.09
kur							*0.26*	*0.27*	0.19	0.19	*0.22*	0.04
I_{Gin}								**0.93**	**0.88**	**0.87**	**0.91**	**0.47**
I_{The}									**0.91**	**0.90**	**0.95**	**0.47**
I_{MLD}										**0.86**	**0.95**	**0.49**
I_{Hoo}											**0.89**	**0.48**
I_{Atk}												**0.49**

5.4 Threats to validity

The results presented above should be considered preliminary and a number of threats to validity should be addressed in the future.

First of all, with respect to construction validity we need to consider a representative set of benchmarks rather than only ArgoUML, Adempiere, and Mogwai, and a representative set of their versions.

Furthermore, our information about the defects might be incomplete, since not all defects might be recorded in the issue tracking system. Moreover, our mapping of defects to classes might be imperfect due to limited recording of this information in the commit messages.

Finally, we have considered only one metric, namely SLOC, and it is not clear whether the results obtained can be generalized to additional metrics.

5.5 Conclusions

As a result of the pilot study, we have observed that the choice of aggregation technique influences the correlation between SLOC values measured per class and aggregated to package level, and the number of defects per package. However, we have observed that bugs which are reported in the issue tracking systems are severely underreferenced in the version control system (VCS) log. For example, only 38 out of the 143 bugs reported and fixed for Mogwai 2.6.0 are referenced in the VCS log. This observation, together with the observation in [9] that only 47.6% of bug fix related commits are documented in the issue tracking systems, i.e., bugs are underreported in the issue tracking systems, lead us to conclude that future studies with the number of defects as validation metric are unfeasible, unless data quality can be ensured.

On the other hand, as a result of the pilot study we have observed high correlation between I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}, and similarly between I_{Kolm}, *mean*, *standard deviation*, and *variance*, suggesting that values aggregated using these techniques convey the same information. However, in the pilot study we have considered only one metric (SLOC), only three systems, and only a single version from each system.

Therefore, we intend to focus on correlation between different aggregation techniques rather than correlation with a validation metric (e.g., number of defects) and we intend to extend the pilot study along the following directions:

Metrics: We extend the study to a number of SLOC variants, i.e., *number of lines of code* (LOC), *number of semicolons* (NOS), and *number of statements* (NOSt), as well as to limited-range metrics such as the *percentage of branching statements* (PBS) and the *percentage of lines with comments* (PLwC) [82], and low variance metrics such as the

depth of inheritance tree (DIT) and the *number of children* (NOC) [22, 108].

Representative set of benchmarks: We extend the study to the 106 systems comprising the Qualitas Corpus. The Qualitas Corpus [110] is a curated collection of open-source Java software systems, intended to be used for empirical studies of code artifacts.

Representative set of versions: We extend the study to the Evolution Corpus, a subset of 13 systems (out of the 106 systems in the Qualitas Corpus) with 10 or more versions available, totaling 414 versions.

Threshold-based techniques: We extend the study by also aggregating metrics from method to class level rather than from class to package level, thus enabling the comparison with the threshold-based aggregation techniques (i.e., SIG and Squale), for which translation functions (from metrics values to individual marks) and thresholds for *source lines of code* measured per method have been published [5, 6, 76].

In order to accommodate these extensions and support the empirical evaluation of different aggregation techniques, appropriate tooling must be developed. The tooling should have following characteristics:

- Flexibility: the tooling should be easily extendible to new metrics and new aggregation techniques, and should permit the aggregation of metrics values at different aggregation levels.
- Integration with metrics-extraction tools: the tooling should enable the further processing of metrics-data previously extracted by third-party tools.
- Reuse of existing components: whenever appropriate, the tooling should reuse existing components, e.g., implementations of the inequality indices or other statistical computations.
- Scalability: the tooling should be robust and scalable with respect to new aggregation techniques, new software systems to be analyzed, and new metrics, both in terms of running time, as well as used resources.
- Export: the tooling should enable the export of processed data to common formats such as CSV.
- Research prototype: the tooling is meant as a research prototype, and its sole user is the researcher performing the experimental evaluation of different aggregation techniques. Therefore, user-interface and portability concerns are superfluous.

Chapter 6

Extensive correlation study

6.1 Introduction

In this chapter we present the results of an extensive correlation study of the traditional and econometric aggregation techniques discussed in Chapters 2 and 3, respectively, applied to lifting metrics values from class to package level. This study represents an extension along two directions of the pilot studies in Chapter 5. First, we consider several other metrics besides *number of source lines of code* (SLOC), i.e., *number of lines of code* (LOC), *number of semicolons* (NOS), *number of statements* (NOSt), *percentage of branching statements* (PBS), *percentage of lines with comments* (PLwC) [82], *depth of inheritance tree* (DIT), and *number of children* (NOC) [22,108]. Second, we consider a representative set of benchmarks, i.e., the 106 systems comprising the Qualitas Corpus [110].

Moreover, apart from measuring the strength of the correlation between the various aggregation techniques, we also investigate the nature of this relation, and we study its evolution. Specifically, we address the following questions:

1. Which and to what extent do the inequality indices *agree*? Which and to what extent do the aggregation techniques rank distributions of metrics values similarly?

2. What is the *nature of the relation* between the various aggregation techniques, i.e., does the scatter plot of the relation exhibit a clear shape (e.g., linear or exponential)?

3. How does the relation between the various aggregation techniques *change with different metrics*?

4. How does the relation between the various aggregation techniques *change in time*, i.e., how does the correlation coefficient change as the systems evolve?

6.2 Methodology

To perform empirical evaluation of different aggregation techniques we conducted two series of experiments. In the first set of experiments (Section 6.3) we investigated relation between pairs of aggregation techniques, i.e., we addressed Questions 1 and 2. As case studies we chose the 106 open-source Java systems comprising the Qualitas Corpus version 20101126r (Section 6.2.1). For each case study we determined the metrics data (SLOC, LOC, NOSt, PBS, PLwC, DIT, NOC) and aggregated it from class level to package level using all pairs of aggregation techniques. As with the pilot studies, we stress that statistical correlations are not transitive [62], i.e., we have to consider *all pairs* of aggregation techniques. By considering different metrics we address Question 3.

An obvious threat to validity for such a study is the representativeness of the versions considered. In order to reduce this threat and to address Question 4, we performed a second set of experiments (Section 6.4), in which we investigated the evolution of the correlation between similar pairs of metrics data sets, again aggregated from class to package level using all combinations of aggregation techniques. As case studies we chose 12 open-source Java systems with more than 10 versions, which are part of the Qualitas Corpus version 20101126e (Section 6.2.1).

6.2.1 Qualitas Corpus Dataset

The Qualitas Corpus [110] is a curated collection of open-source Java software systems, intended to be used for empirical studies of code artifacts.

We consider the Corpus version 20101126[1], which comes in two main distributions. For our first study (Section 6.3) we consider the "r" (recent) variant, which contains the most recent versions available at the time of release, from 106 systems ranging from *FitJava v1.1* (2 packages, 2240 SLOC) to *NetBeans v6.9.1* (3373 packages, 1890536 SLOC). The largest three systems (in terms of number of packages) are *NetBeans v6.9.1* (3373 packages, 1890536 SLOC), *Eclipse SDK v3.6* (1313 packages, 2282511 SLOC), and *JRE v1.6.0* (1144 packages, 914867 SLOC). Similarly, the smallest three systems are *Squirrel SQL v3.1.2* (3 packages, 6944 SLOC), *Trove v2.1.0* (3 packages, 2196 SLOC), and *FitJava v1.1* (2 packages, 2240 SLOC).

Our second study uses the "e" (evolution) variant of the Qualitas Corpus 20101126, which contains all versions from 13 systems (out of the 106 systems) with 10 or more versions available, totaling 414 versions. We have excluded *Eclipse SDK* (represented by 35 versions) from the consideration, because for 34 out of the 35 versions there is only bytecode available, while we focus on source code metrics (e.g., SLOC). All other systems had the

[1]Qualitas Research Group, Qualitas Corpus Version 20101126, `http://qualitascorpus.com`. The University of Auckland, February 2009.

source code available for all versions. Hereafter we refer to the remaining twelve systems as the Evolution Corpus.

The most covered systems of the Evolution Corpus in terms of number of versions available are *Hibernate* (86 versions), *Azureus/Vuze* (51 versions), and *Weka* (49 versions), while the least covered three systems are *ArgoUML* (10 versions), *ANTLR* (18 versions), and *JMeter* (18 versions). In terms of size (in terms of number of packages), the Evolution Corpus ranges between 634 packages in *Hibernate v3.6.0-beta4* and 6 packages in *Ant v1.1.*

6.2.2 Data collection

For both the single snapshot study, as well as the evolutionary study, the source code for each version of each system was automatically processed, then the package structure (i.e., the list of packages and the classes contained in each package) and the metrics data were extracted. Metrics data for LOC could have also been extracted from the Qualitas Corpus metadata. However, we preferred to extract all metrics data using our own tooling as it was reported in the release notes of the 20101126 version of the Corpus [109] that some of the previous metadata files contained incorrectly-computed values.

An important note is the distinction between source code of the actual system, and source code of third-party libraries. It is possible that some systems distribute original source code of third-party libraries (in a previous release of the Corpus [110] it was reported that, e.g., *Compiere* v250 distributes a copy of the source code of the Apache Element Construction Set), while others provide their own implementations of such libraries, i.e., they distribute modified versions of third-party libraries together with their own source code.

We focus on source code of the actual systems and we exclude libraries. The decision regarding what is identified as actual source code of a version, and what is considered third-party is documented and provided as metadata alongside the Corpus, i.e., a space-separated list of prefixes of packages of Java types which are considered as developed for the system. For example, for *ArgoUML* all packages with names prefixed by `org.argouml` are considered as source packages (e.g., `org.argouml.model`), while all others are considered as externals (e.g., `org.apache.xerces`).

For all pairs of aggregation techniques compared, we considered packages containing at least 2 classes. This is motivated by the fact that, when applied to packages containing only one class, most of the traditional aggregation techniques (standard deviation, variance, skewness, and kurtosis) are undefined, and all inequality indices are equal to 0. For Qualitas Corpus 20101126r, the most affected systems by this filtering were some of the small ones, namely *IvataGroupware* v0.11.3 (lost 34 out of 81 packages), *Sandmark* v3.4 (lost 45 out of 123 packages), and Quilt v0.6-a-5 (lost 5 out

of 14 packages). Across the entire Corpus 20101126r, 86.7% of all systems lost less than 20% of their packages.

For each package in each version of each system (in both studies), we aggregate the metrics values (SLOC, LOC, NOSt, PBS, PLwC, DIT, NOC) of all the classes *directly* contained in that package, in turn, using each of the aggregation techniques considered. Recall from Section 5.2.1 that we say that a class C is directly contained in a package P if there exists no subpackage P' of P different from P such that C is contained in P'.

6.2.3 Data analysis

As discussed in the data analysis methodology of the pilot studies (Section 5.2.2), we opt for Kendall's rank correlation coefficient τ [57] to measure correlation between values aggregated using different techniques. All computations were performed using R [92].

Next we describe the two series of experiments performed.

6.3 Studying the correlation between aggregation techniques

In the first series of experiments we study correlation between the aggregation techniques on a single snapshot of each system in the Corpus, and we answer Questions 1, 2, and 3.

6.3.1 Which and how much do aggregation techniques agree?

Agreement between inequality indices

We start by measuring the Kendall correlation between metrics values aggregated using the inequality indices (Figures 6.1-6.3) across the entire Corpus. Note that the percentage of the systems in the Corpus for which the correlation value is statistically significant using the common threshold of 0.05 is displayed between parentheses below each boxplot, the metric is displayed on the left, and the name of each aggregation technique is abbreviated to its first three letters. Moreover, we display horizontal dotted lines to indicate the median for each of the boxplots. To simplify comparison of different plots, we opt for the same scale in all figures.

We observe high and statistically significant correlation between I_{Theil}, I_{MLD}, I_{Gini}, I_{Atkinson}, and I_{Hoover} in more than 90% of the Corpus for all metrics, i.e., aggregation values obtained using these techniques convey the same information. This confirms our observation in the pilot studies in Chapter 5 and answers the first part of Question 1 from Section 6.1: I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover} and I_{Atkinson} agree regardless of the metric, i.e., they

rank distributions of metrics values similarly (there is high and statistically significant correlation between them).

Correlation between I_{Kolm} and the other indices (Figure 6.3) is average at best (0.4–0.5), and significant for approximately 70% of the Corpus for all metrics except NOC and PBS. For NOC and PBS, the correlation between I_{Kolm} and the other inequality indices is negative, and much less statistically significant, i.e., approximately 20% of the Corpus for NOC, and approximately 40% of the Corpus for PBS.

Agreement between other aggregation techniques

Next we study how the other aggregation techniques correlate to each other and to the inequality indices. Since I_{Kolm} did not show high correlation with any of the inequality indices, we also study whether and to which traditional techniques I_{Kolm} correlates more. Recall that during the pilot studies in Chapter 5 we observed high and statistically significant correlation between I_{Kolm} and *mean*, *standard deviation*, and *variance*.

In Figures 6.4 and 6.5, *mean* shows low positive correlation (0.2–0.3) with all the inequality indices except I_{Kolm} (significant in approximately 50% of the Corpus), for SLOC, LOC, NOS, and NOSt, i.e., all the size metrics considered. In contrast, *mean* shows low-average negative correlation (0.4–0.5) with all the inequality indices except I_{Kolm}, for DIT, NOC, PBS, and PLwC.

The correlation between *mean* and I_{Kolm} (Figure 6.4) is the highest (0.8) among all other techniques (statistically significant for 92% of the systems), again for SLOC, LOC, NOS, and NOSt, i.e., aggregates obtained using these techniques convey the same information. The low-variance metrics provide contradictory results (Figure 6.5), i.e., for NOC correlation between *mean* and I_{Kolm} is the highest (0.8) among all other techniques, while for DIT correlation between *mean* and *median* is the highest (0.8) among all other techniques, with correlation between *mean* and I_{Kolm} dropping below average (0.4), and being statistically significant for only 51% of the Corpus. For the limited range metrics, i.e., PBS and PLwC, correlation between *mean* and *median* is the highest (0.8) among all other techniques (statistically significant for 90% of the Corpus).

Moreover, for the size metrics, *mean* shows positive high (0.7–0.8) and statistically significant correlation for 80–90% of the Corpus with *median*, *standard deviation*, and *variance*. In contrast, for the low-variance and limited-range metrics, correlation between *mean*, *median*, *standard deviation*, and *variance* is weaker and less statistically significant.

Correlation between I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}

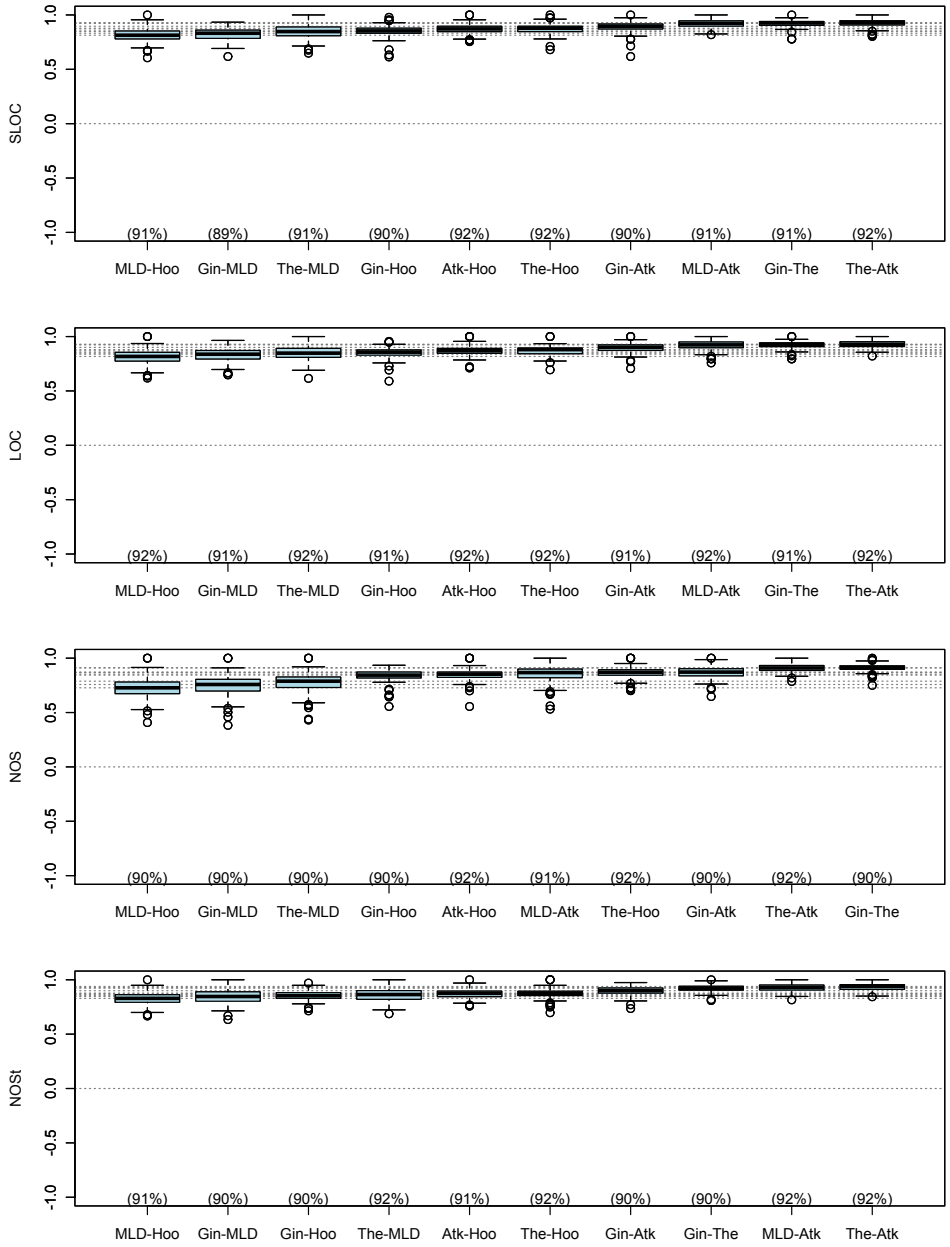

Figure 6.1: All inequality indices except I_{Kolm} show high correlation with each other for size metrics.

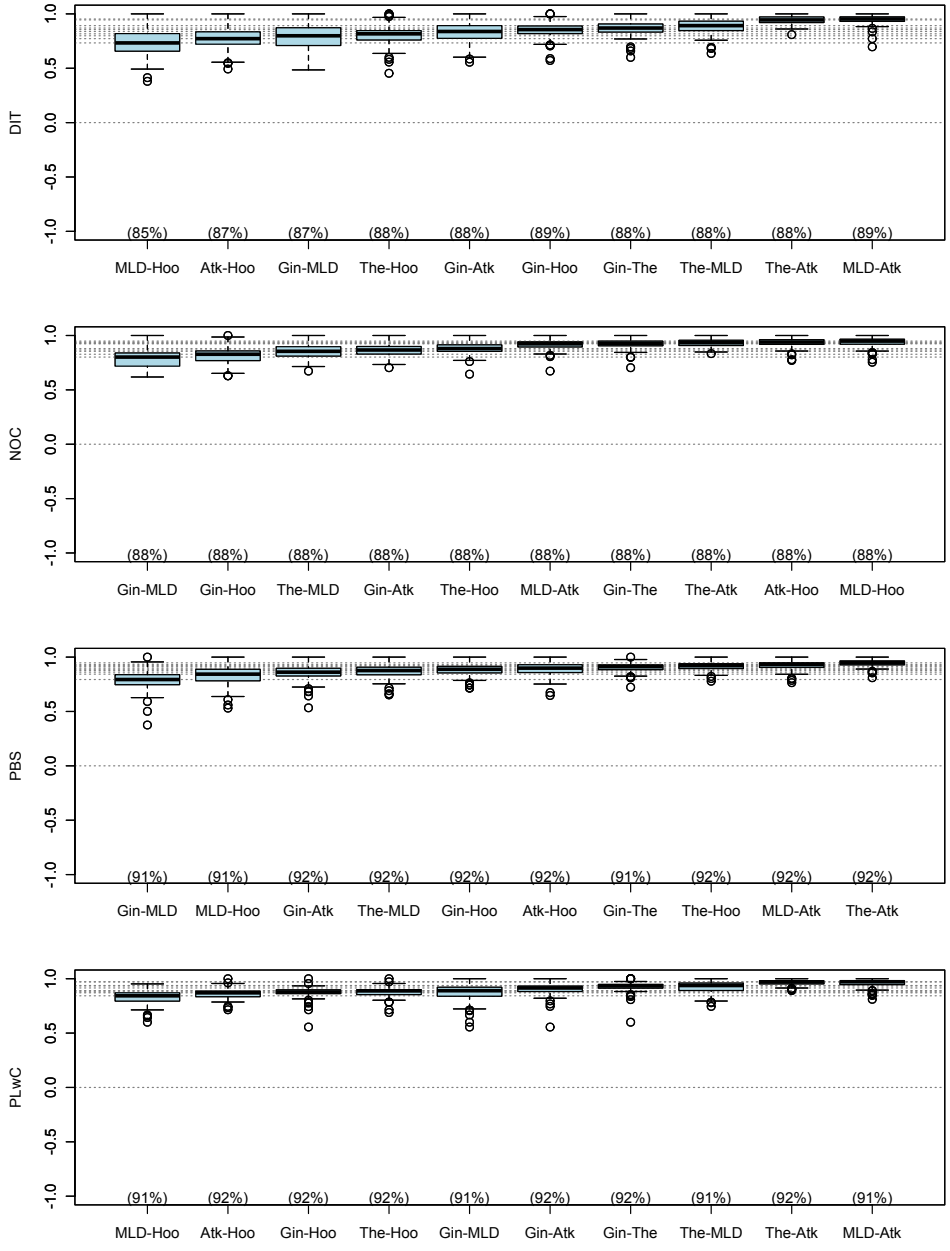

Figure 6.2: All inequality indices except I_{Kolm} show high correlation with each other for low-variance metrics and limited-range metrics.

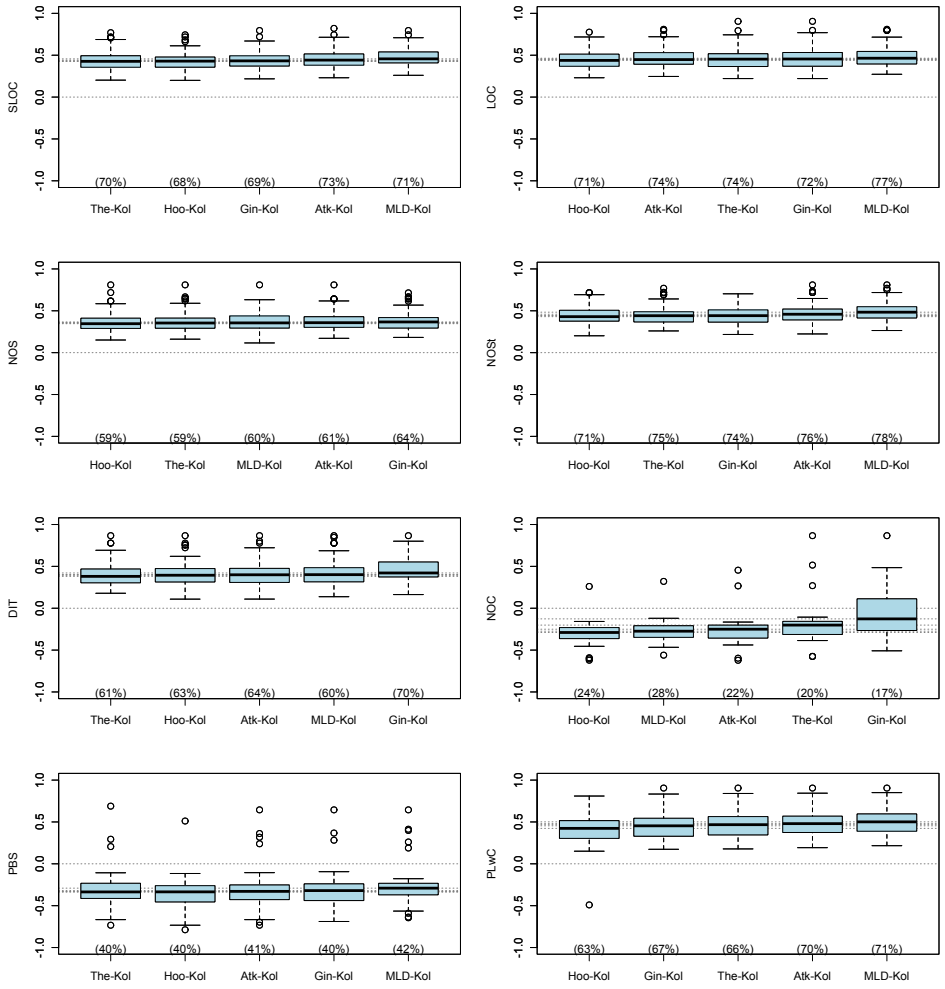

Figure 6.3: I_{Kolm} does not correlate with other inequality indices.

Closer inspection at I_{Kolm} (Figures 6.6) reveals high (0.8) and statistically significant correlation in 90% of the Corpus with *mean*, *standard deviation*, and *variance* for the size metrics. It is interesting to observe that while *mean* shows high correlation with both *median* (0.7) and I_{Kolm} (0.8), the correlation between *median* and I_{Kolm} is lower (0.5–0.6). For the low variance metrics DIT and NOC (Figure 6.7), strong and statistically significant correlation between I_{Kolm} and *standard deviation* and *variance* can also be observed in 80–90% of the Corpus. However, *mean* is inconsistent: while there high and statistically significant correlation between *mean* and I_{Kolm} in 86% of the Corpus for NOC, the same correlation for DIT is below average (0.4), and is much less statistically significant (51% of the Corpus). For the limited range metrics PBS and PLwC, *standard deviation* and *variance* correlate the most with I_{Kolm}, while *mean* is again inconsistent.

Correlation between *mean* and other aggregation techniques

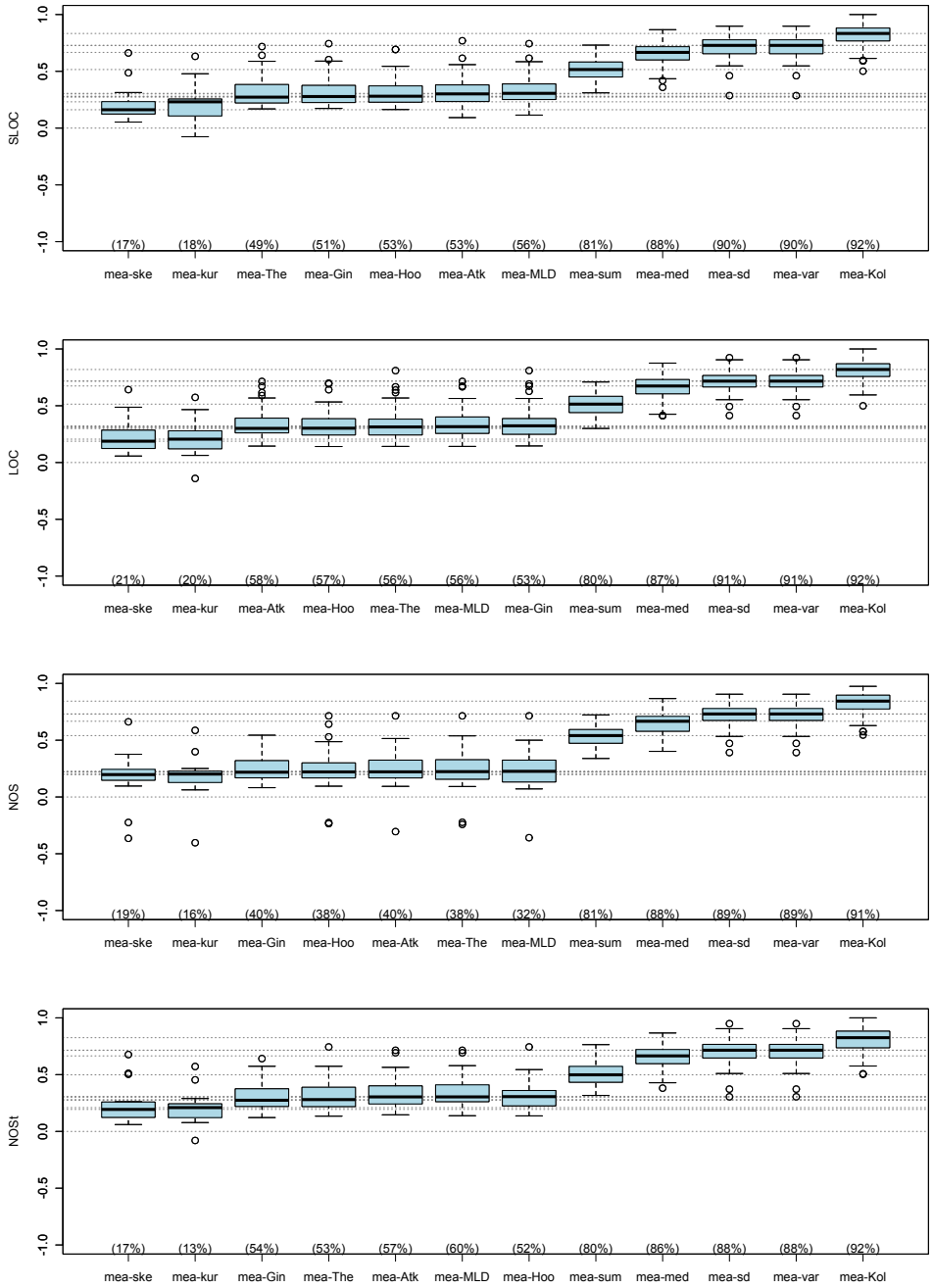

Figure 6.4: *Mean* shows high correlation with I_{Kolm} for size metrics.

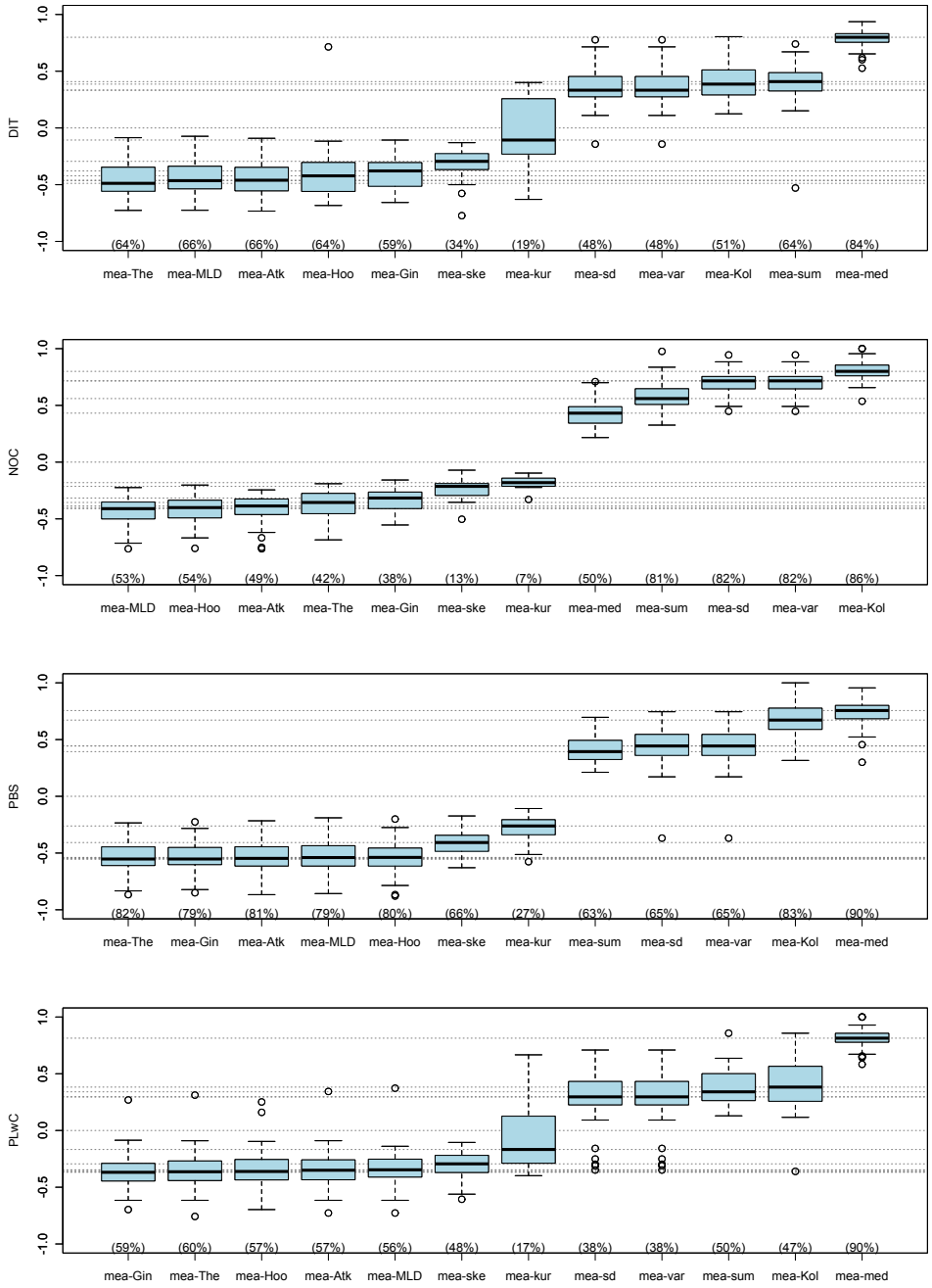

Figure 6.5: *Mean* shows high correlation with either *median* or I_{Kolm} for low-variance metrics and limited-range metrics.

Correlation between I_{Kolm} and other aggregation techniques

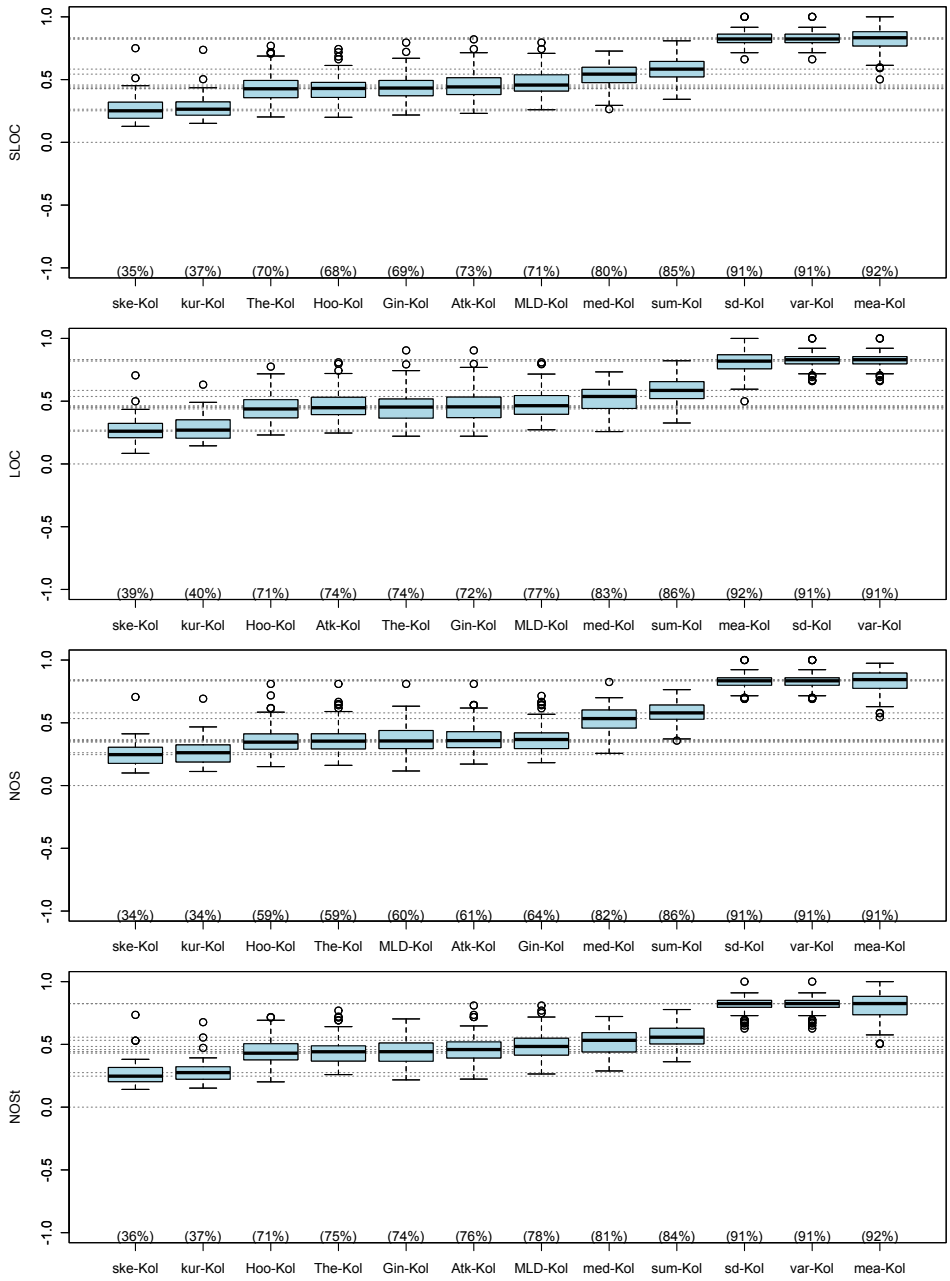

Figure 6.6: I_{Kolm} shows high correlation with *mean*, *standard deviation*, and *variance* for size metrics.

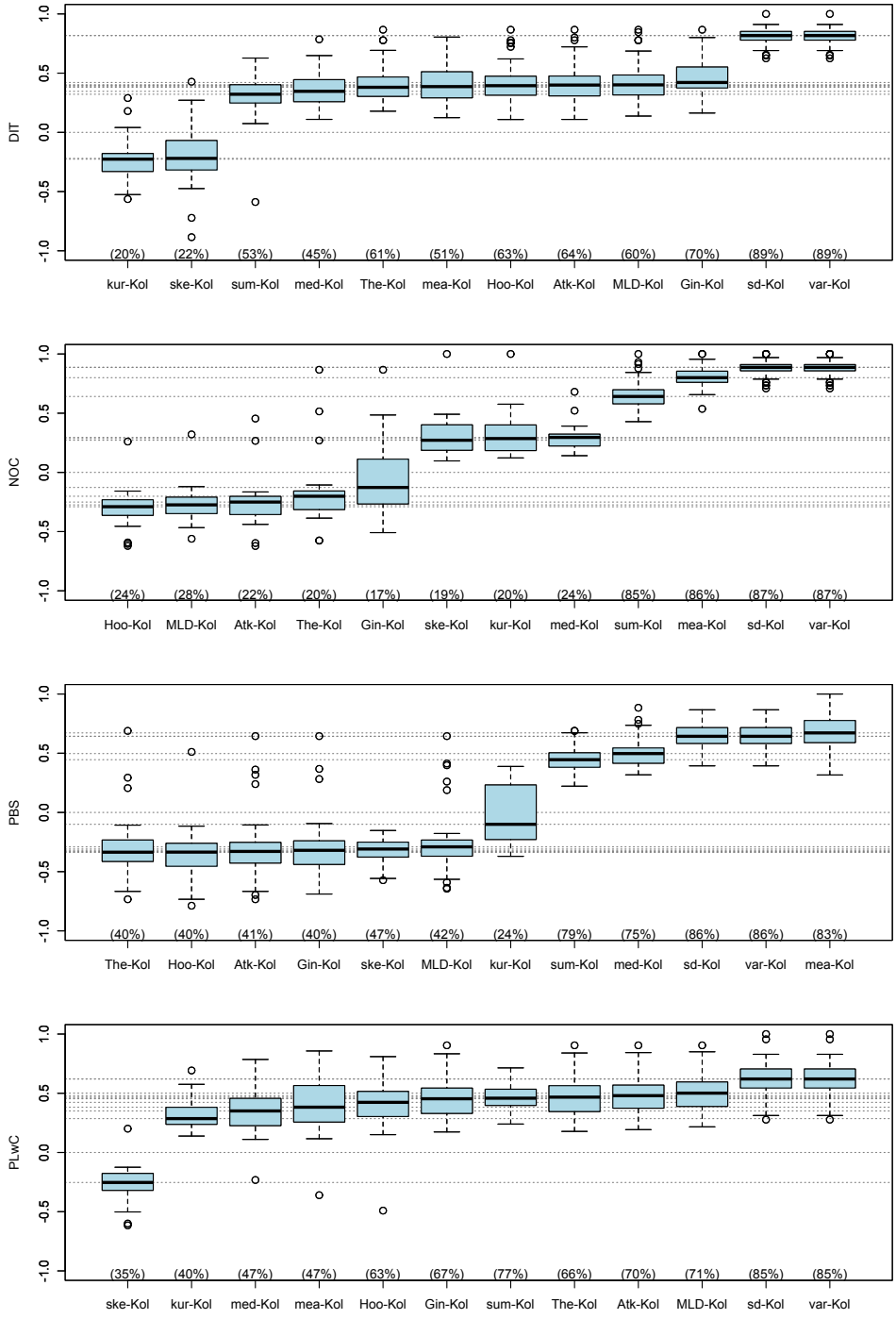

Figure 6.7: I_{Kolm} shows high correlation with *standard deviation* and *variance* for low-variance metrics.

Correlation between *sum* and other aggregation techniques

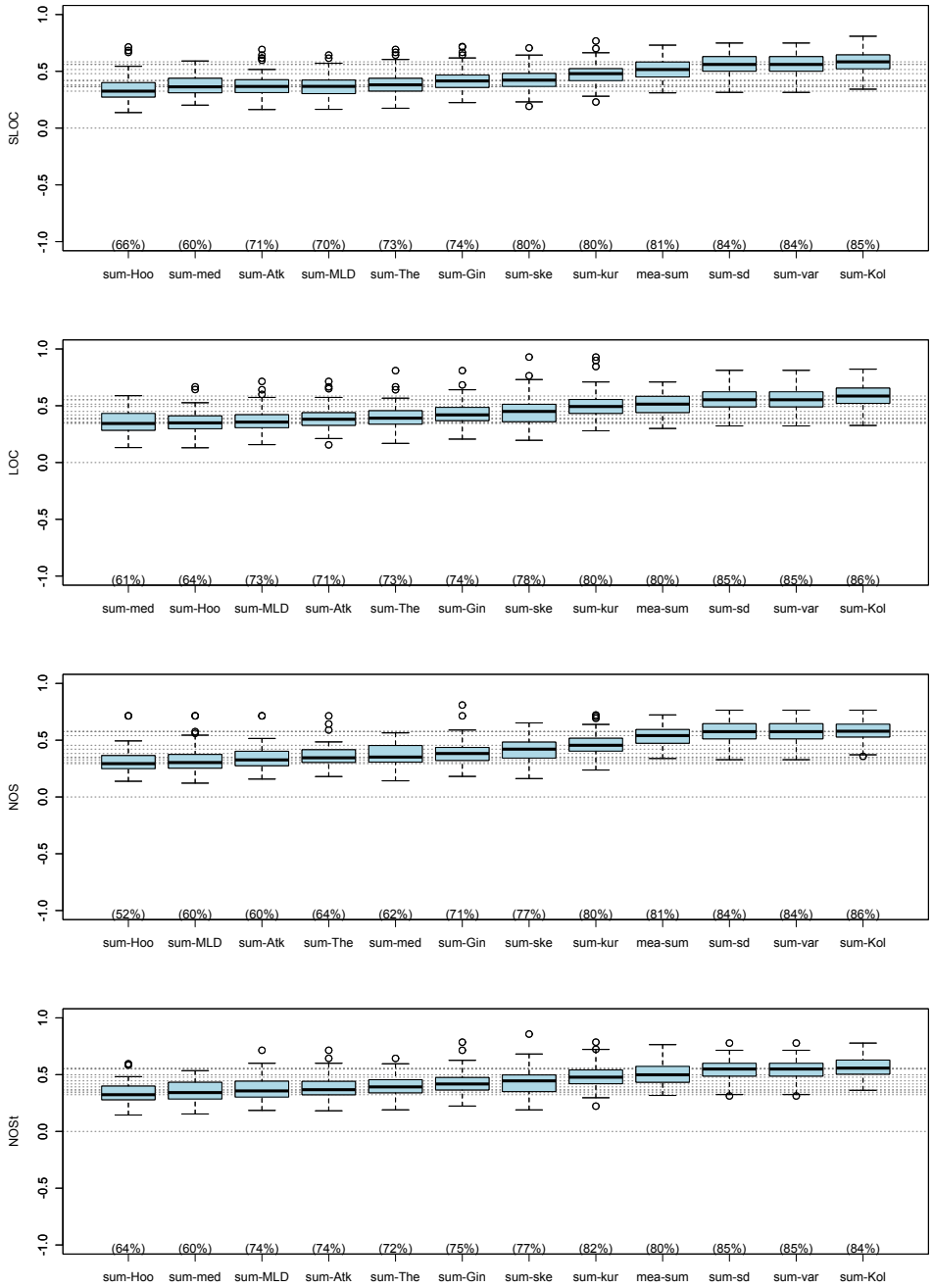

Figure 6.8: *Sum* does not correlate to the other aggregation techniques for size metrics.

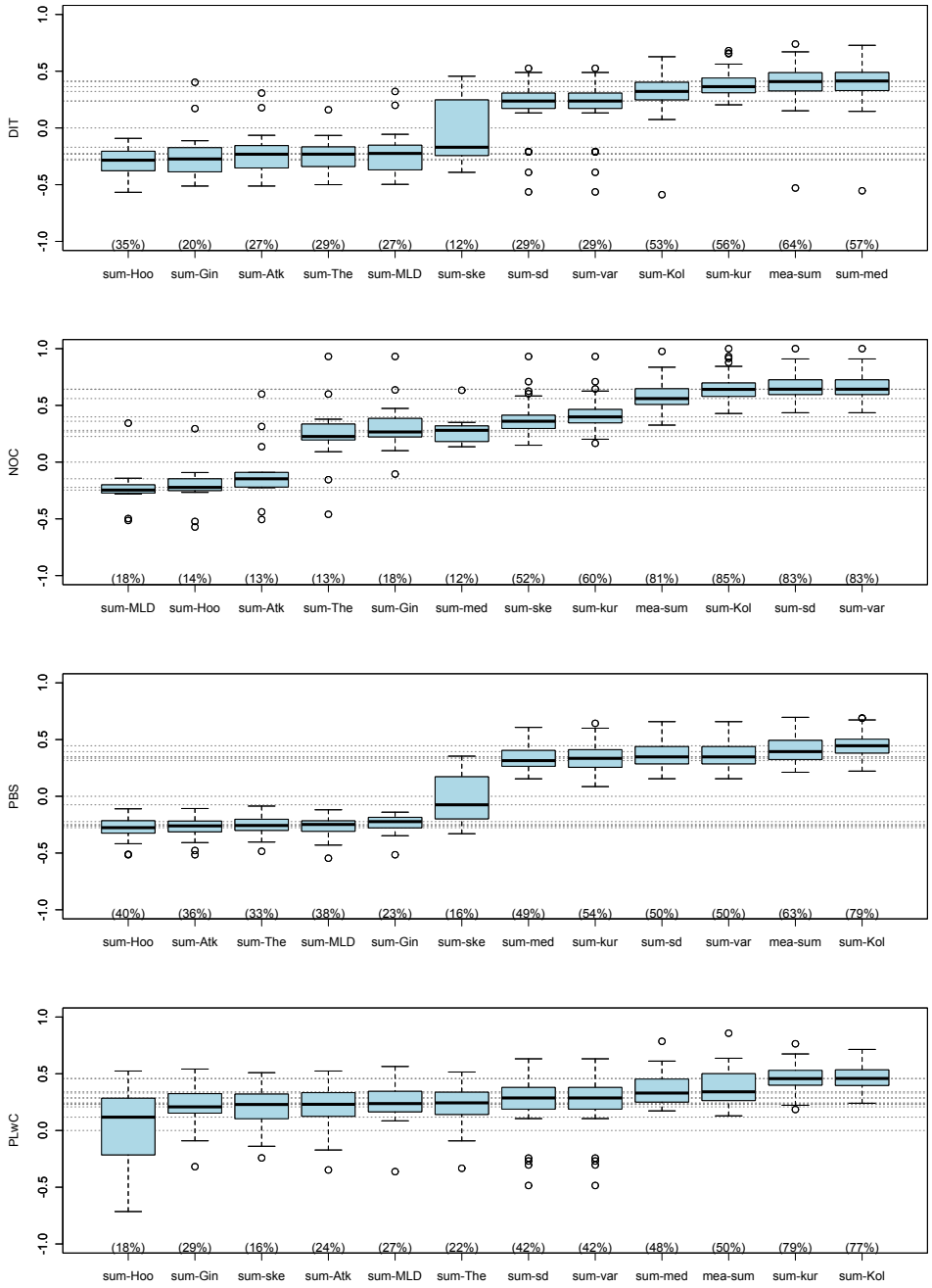

Figure 6.9: *Sum* does not correlate to the other aggregation techniques for low-variance metrics and limited-range metrics.

The results for *sum* are consistent for the size metrics (Figure 6.8), where the correlation with I_{Kolm} is the strongest among the techniques considered (0.5–0.6) for approximately 80% of the Corpus. For PBS and PLwC (Figure 6.9), the correlation between *sum* and I_{Kolm} is also the strongest among the techniques considered, but it drops below average, while the results for DIT and NOC (Figure 6.9) are inconsistent.

The correlation between *standard deviation* and *variance* is perfect (1), since one is a monotonic transformation of the other, which does not influence the results of the Kendall rank correlation coefficient. Similarly, we observe high correlation (0.8) between *skewness* and *kurtosis* for the size metrics (Figure 6.10). In the other two cases (DIT and NOC on the one hand, and PBS and PLwC on the other hand), *skewness* and *kurtosis* disagree, i.e., the correlation coefficient is at most average (0.5) for DIT, PBS, and PLwC, and it is very high (0.9) for NOC.

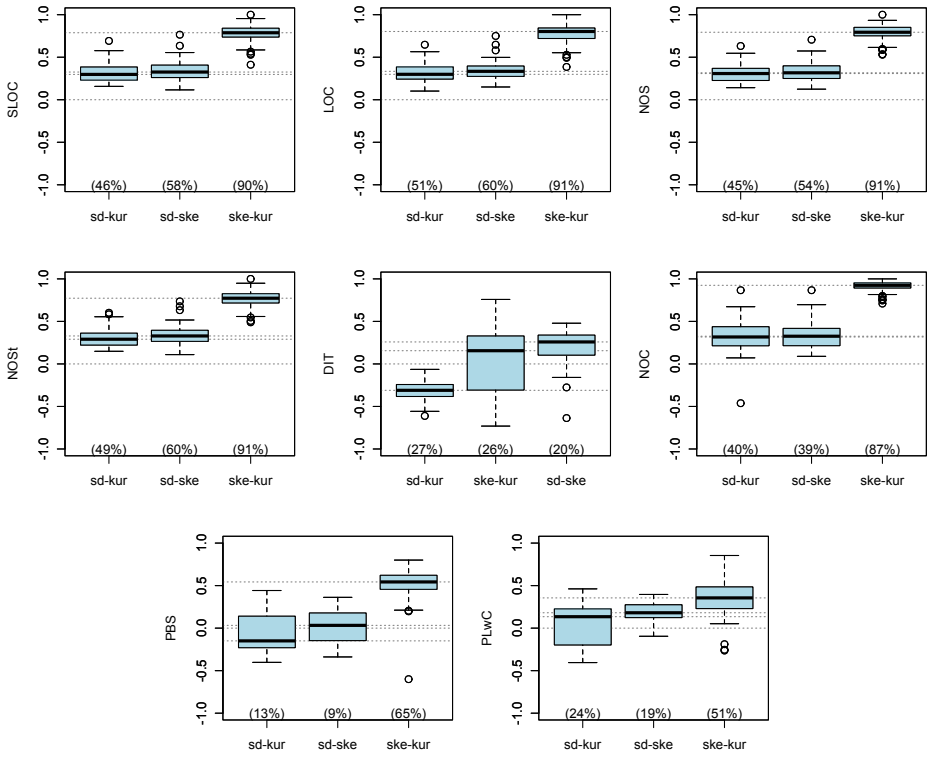

Figure 6.10: Correlation between *standard deviation*, *skewness*, and *kurtosis*.

Hence, to summarize our answer to Question 1, we note that:

- I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover} and I_{Atkinson} show consistently high and statistically significant correlation between them, i.e., the aggregates obtained using these techniques convey the same information, regardless of the metric considered.

- The correlation between *mean* and I_{Kolm} is high and statistically significant for the size metrics, again suggesting that aggregates obtained using *mean* and I_{Kolm} convey the same information. Moreover, *mean* shows high and statistically significant correlation with *median*, *standard deviation*, and *variance* for the LOC-based metrics. However, *mean* is inconsistent for the low variance metrics and the limited range metrics, i.e., the correlation between *mean* and the other techniques depends on the choice of metric.

- Aggregated values obtained using *standard deviation* or *variance* convey the same information for all metrics, while values obtained using *skewness* and *kurtosis* convey the same information for the size metrics.

6.3.2 What is the nature of the relation between aggregation techniques?

In order to study the nature of these relations, we draw scatter plots for each pair of aggregation techniques and each system in the Corpus and we analyze if the scatter plots exhibit a clear shape. In particular, we are interested in observing linear, superlinear, or chaotic patterns, although the Kendall *rank* correlation values previously computed are not sensitive to the linearity of these relations. To illustrate the relation between metrics values aggregated using different aggregation techniques we choose *Compiere* as a representative example[2].

Nature of relation between I_{Gini}, I_{Theil}, I_{Hoover}, I_{Atkinson}, and I_{Kolm}

To illustrate the relation between metrics values aggregated using the inequality indices for each metric considered we refer to Figures 6.11–6.18. We distinguish a clear linear relation between I_{Theil} and I_{Atkinson} whenever the maximal inequality is low, hence I_{Atkinson} does not cover its entire possible range $\left[0, 1 - \frac{1}{n}\right]$, where n is the number of values being aggregated (Figures 6.11–6.15, 6.18). In contrast, when I_{Atkinson} ranges over the entire interval $\left[0, 1 - \frac{1}{n}\right]$ (Figures 6.16 and 6.17), we distinguish a superlinear relation between I_{Theil} and I_{Atkinson}. Regardless of the range of I_{Atkinson} or the

[2]The other systems in the Corpus exhibit similar patterns. For complete results see `www.student.tue.nl/X/b.n.vasilescu/scatterPlots/scatter.html`

exact metric, correlation between I_{Theil} and I_{Atkinson} does not drop below 0.90 and is always statistically significant.

The also high measured correlation between I_{Theil} and I_{Gini}, however, corresponds to a clear relation which visually exhibits superlinear rather than linear growth, regardless of the observed range of I_{Gini}. This observation agrees with the econometric-based distinction between different *dimensions of inequality* [21], and the sensitivity of the different inequality indices to different such dimensions, as discussed in Section 3.2. Our experiments confirm the claim in [21] that I_{Theil} is highly sensitive to inequality associated with the exceptionally rich, since the sharper increase in I_{Theil} as I_{Gini} increases, i.e., as the inequality between the "rich" and the "poor" increases, is visible in the top right plots in Figures 6.11–6.18.

The high correlation values between I_{Theil} and I_{Hoover} are also supported by a relation similar to the one between I_{Theil} and I_{Gini}, which appears visually to be superlinear, although we observe more dispersion, i.e., disagreement, towards the "rich". On the other hand, the chaotic pattern is observed between I_{Theil} and I_{Kolm} (Figures 6.11–6.18).

Fitting general parametric distributions to model the relations between the inequality indices goes beyond the scope of this thesis.

Nature of the relation between other aggregation techniques

Next we study the relation between I_{Kolm} and the two aggregation techniques with which it showed high correlation, i.e., *mean* and *median* (Figures 6.19 and 6.20). We observe a linear relation between *mean* and I_{Kolm} for the size metrics (Figure 6.19), and a much more chaotic relation for the low-variance and limited-range metrics (e.g., see DIT and PLwC in Figure 6.20). The relation between *median* and I_{Kolm}, albeit still visible, is more chaotic both for the size metrics, as well as for low-variance and limited-range metrics.

Finally, the high and statistically significant correlation observed between *skewness* and *kurtosis* for the size metrics is witnessed by a superlinear relation in Figure 6.21. In contrast, the relation between *skewness* and *kurtosis* is more chaotic for the low-variance and limited-range metric, for which the correlation is also very low (with the exception of NOC).

To summarize our answer to Question 2, we note that linear, superlinear, as well as chaotic patterns can be observed in the scatter plots. However, high correlation values do not always correspond to clear-shaped relations (e.g., between I_{Theil} and I_{Hoover} for DIT). We observe that linear or superlinear relations always correspond to high correlation values, while chaotic patterns correspond to both high and average/low correlation values.

Shape of the relation between I_{Gini}, I_{Theil}, I_{Hoover}, I_{Atkinson}, and I_{Kolm}

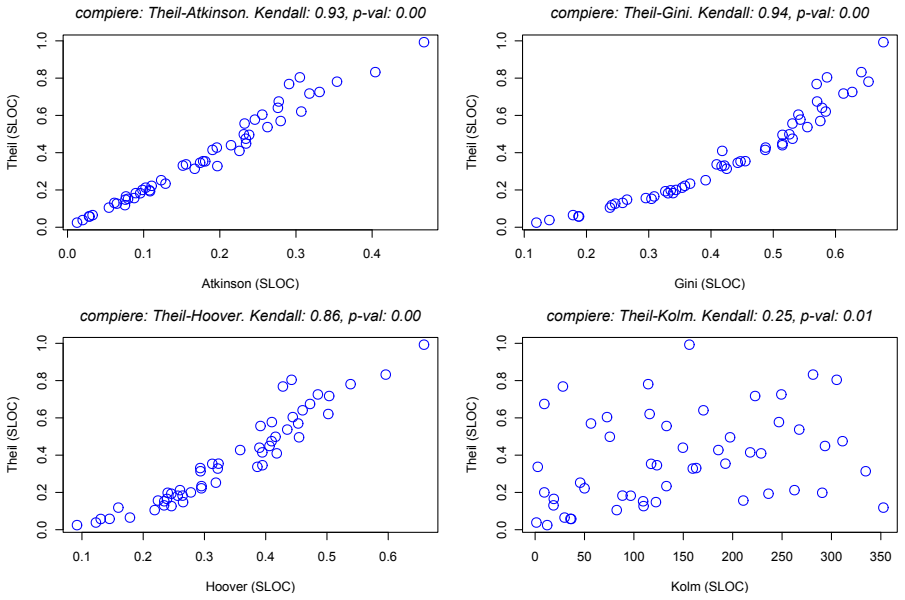

Figure 6.11: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for SLOC.

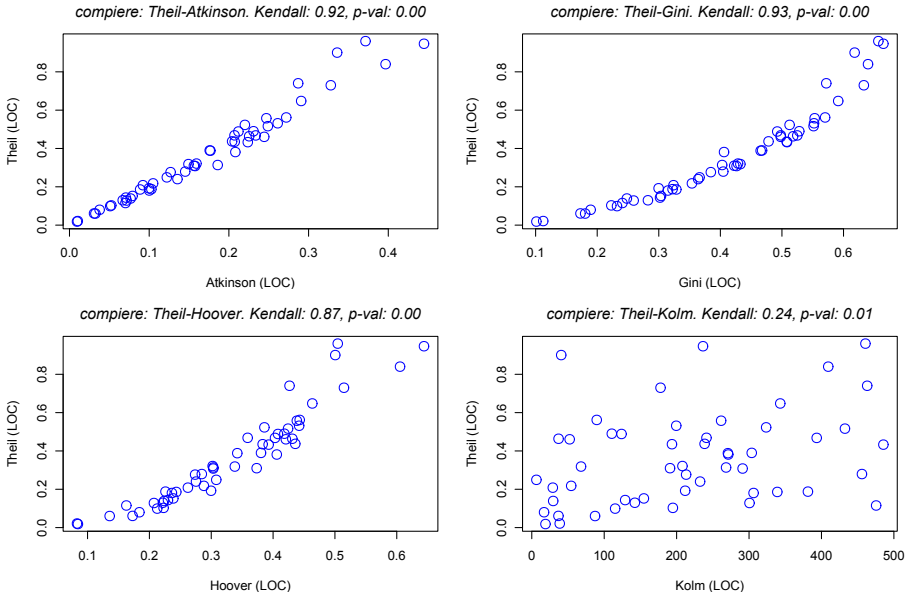

Figure 6.12: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for LOC.

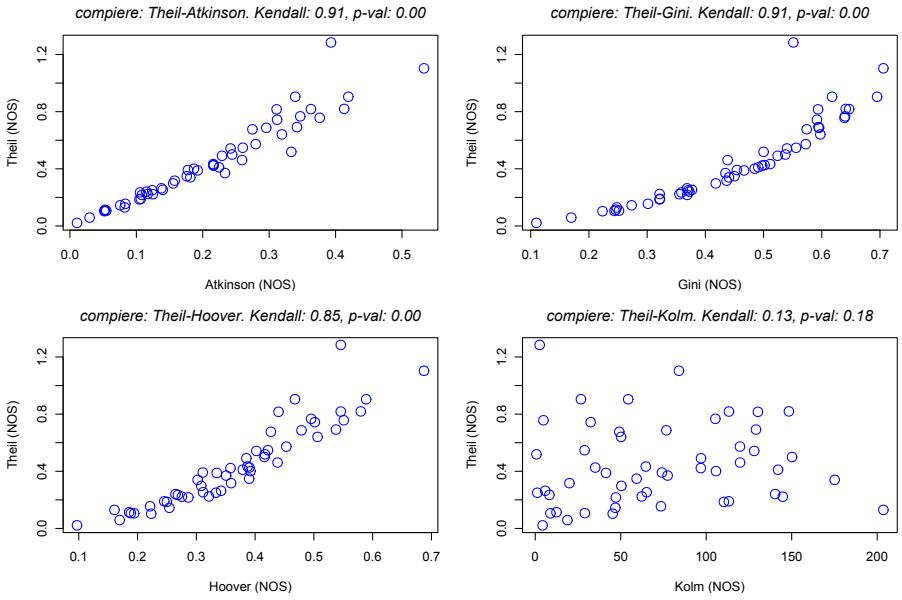

Figure 6.13: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for NOS.

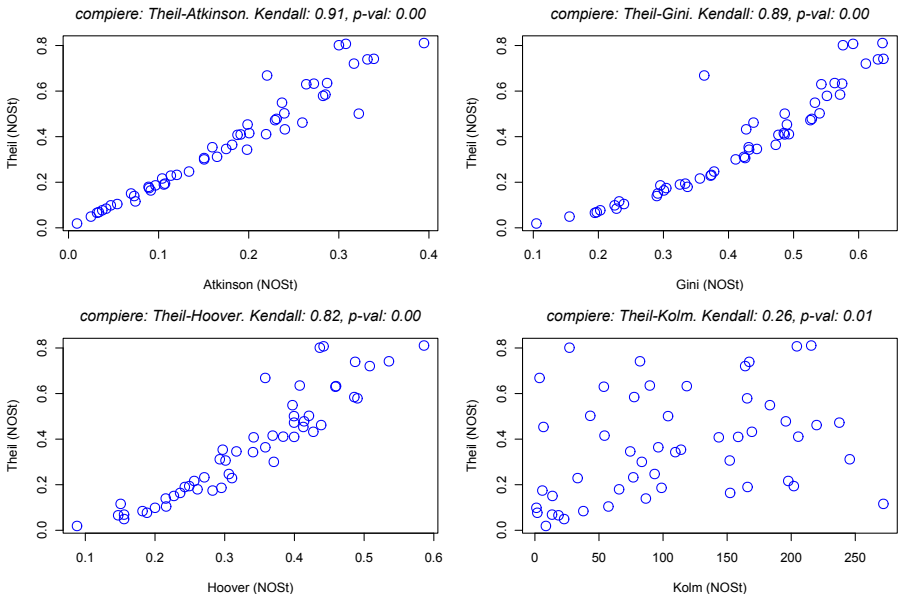

Figure 6.14: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for NOSt.

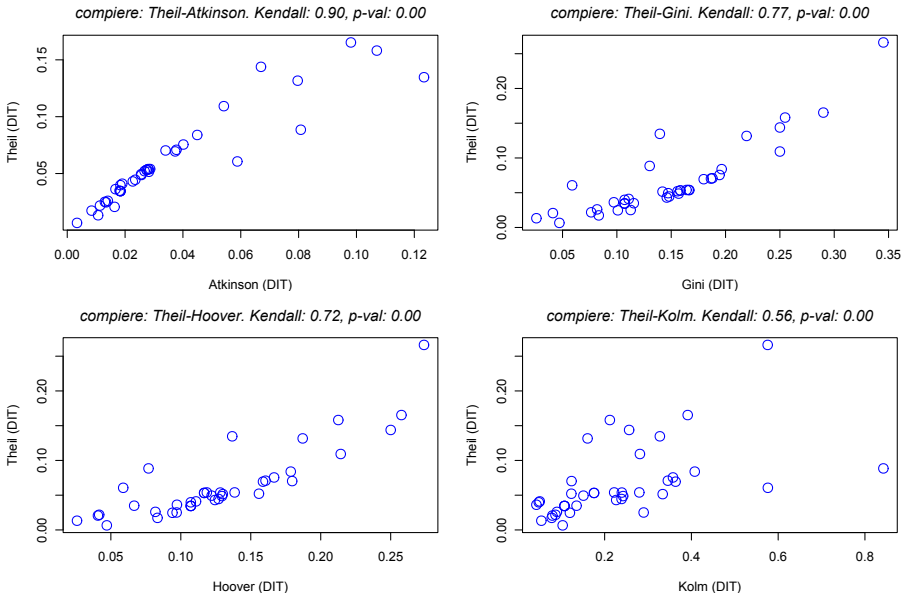

Figure 6.15: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for DIT.

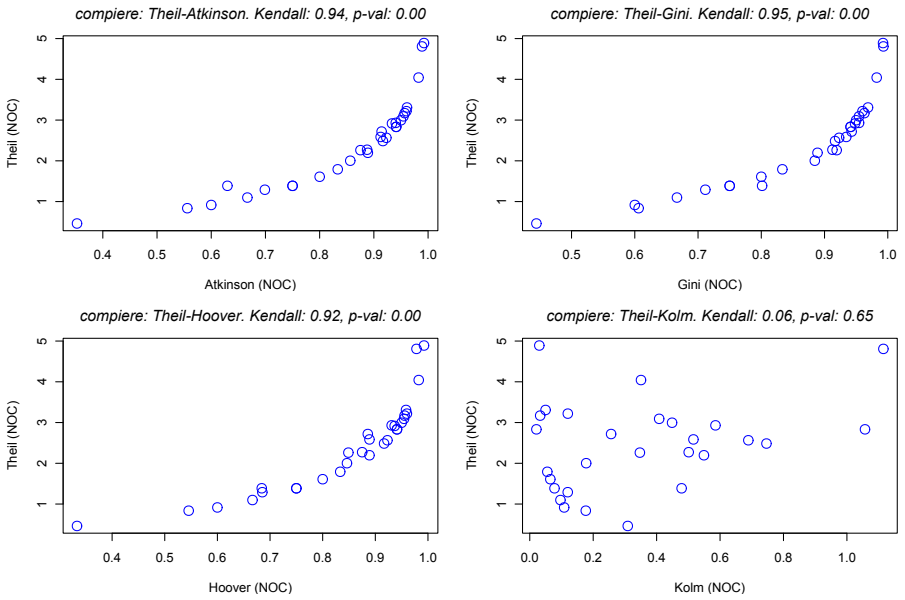

Figure 6.16: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for NOC.

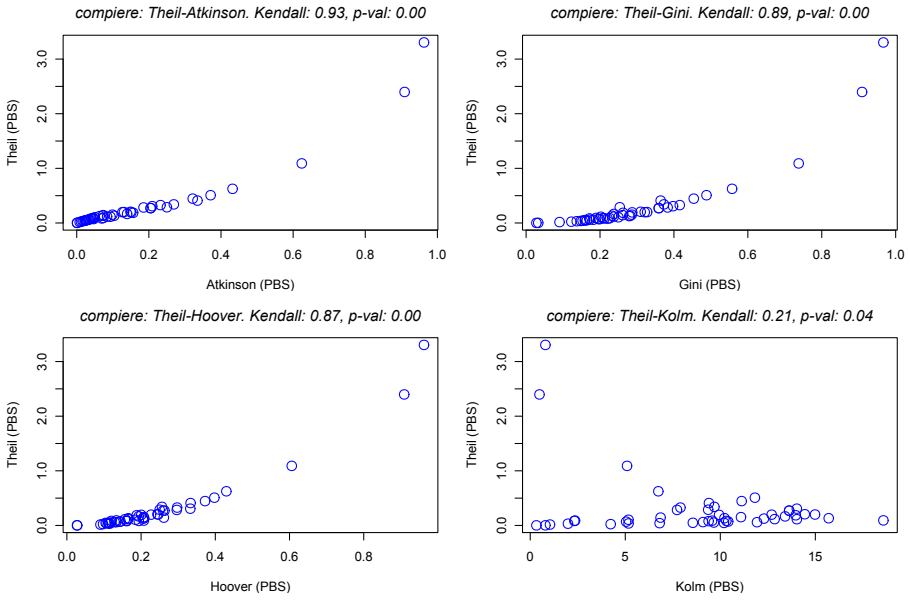

Figure 6.17: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for PBS.

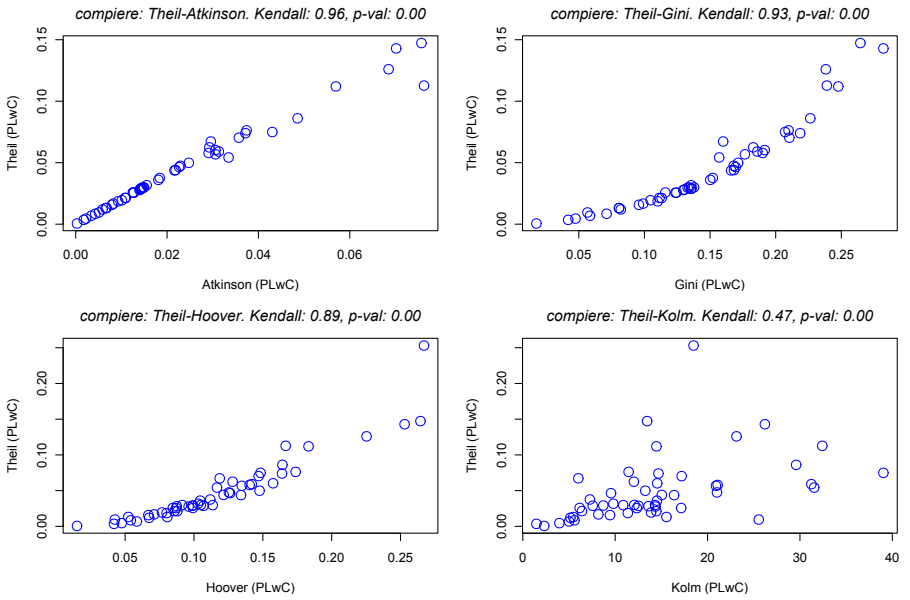

Figure 6.18: Shape of the relation between I_{Theil} and each of I_{Atkinson}, I_{Gini}, I_{Hoover}, and I_{Kolm} for PLwC.

Shape of the relations *mean*–I_{Kolm} and *median*–I_{Kolm}

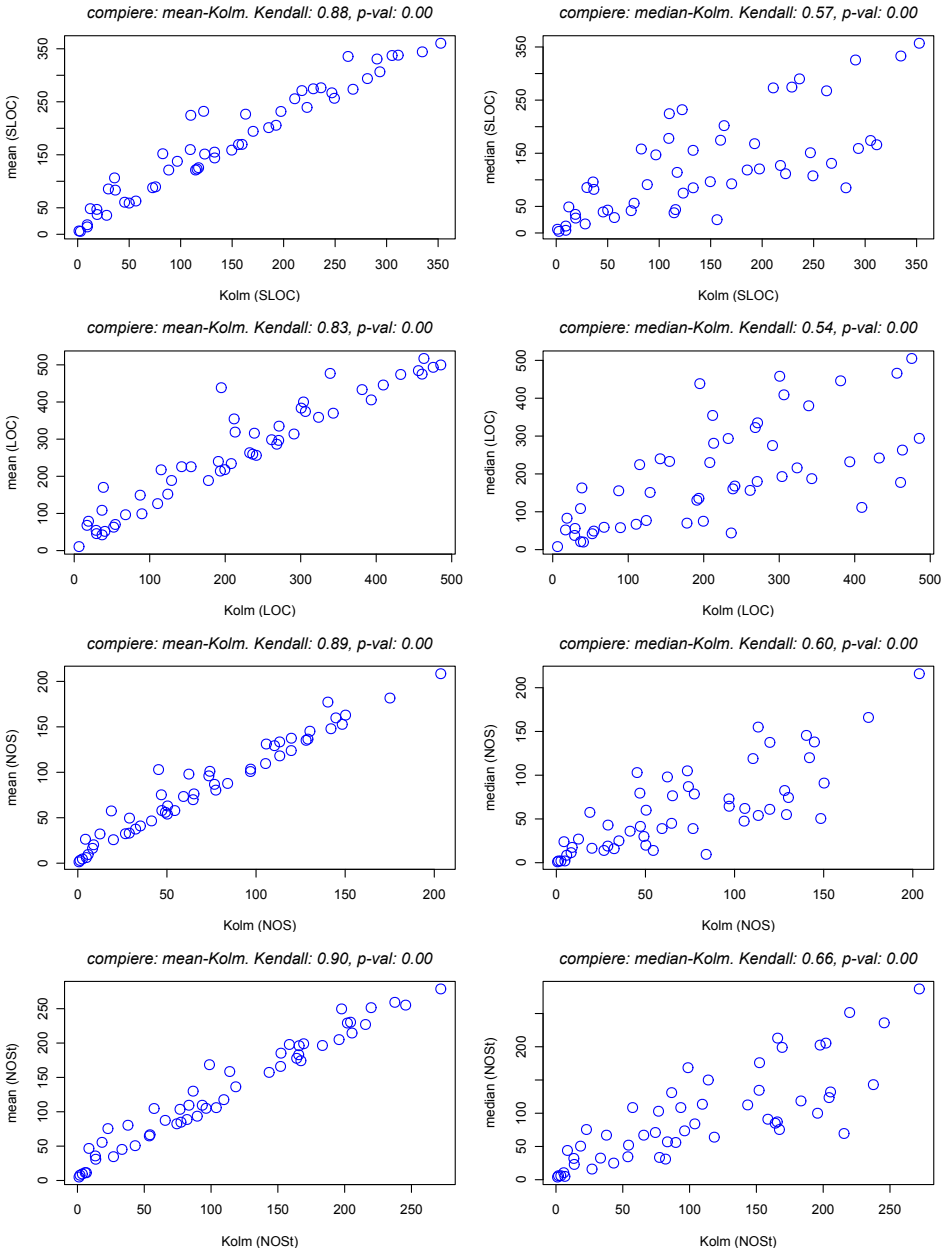

Figure 6.19: Relation between *mean* and I_{Kolm}, and between *median* and I_{Kolm}, for size metrics.

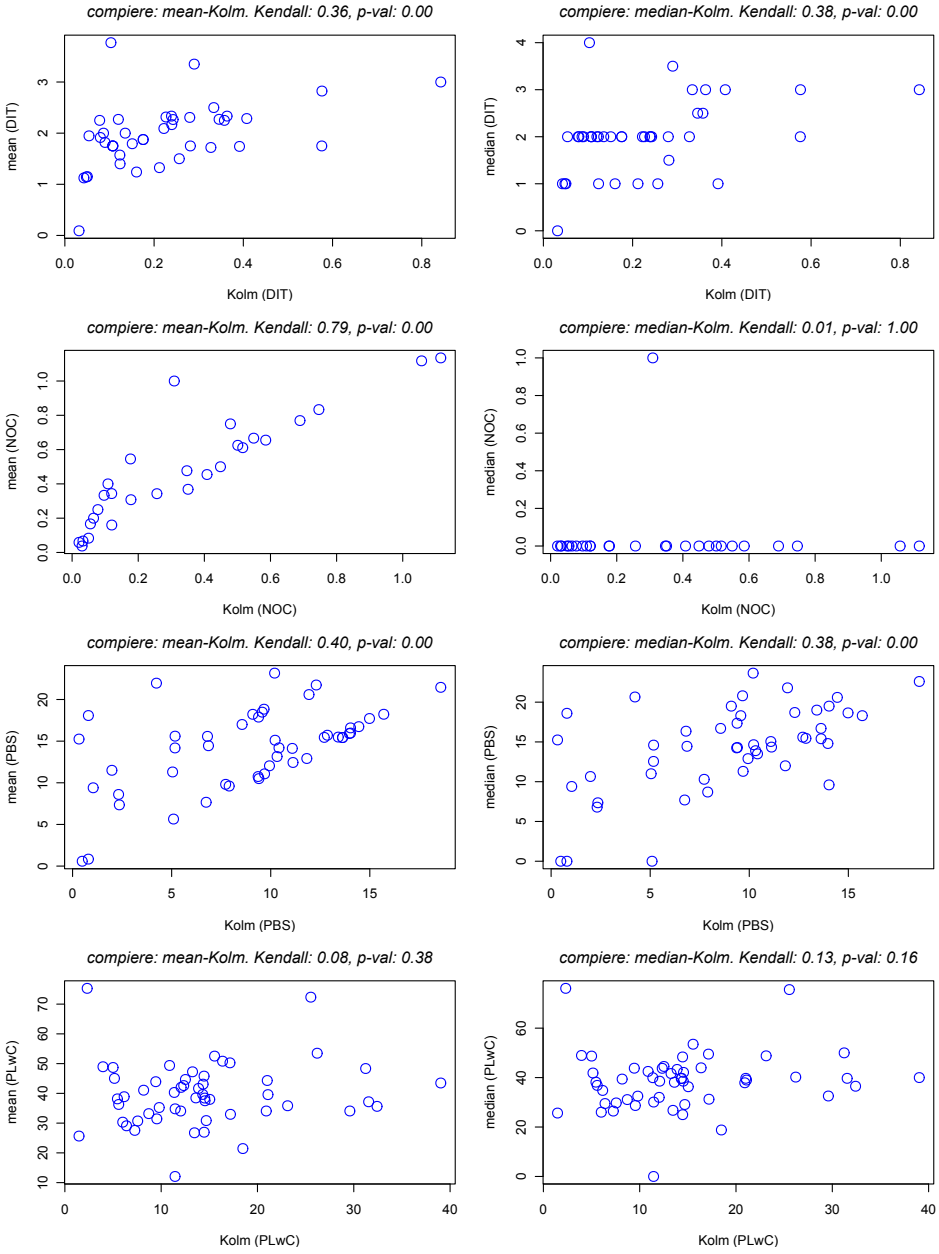

Figure 6.20: Relation between *mean* and I_{Kolm}, and between *median* and I_{Kolm}, for low-variance metrics and limited-range metrics.

Shape of the relation between *skewness* and *kurtosis*

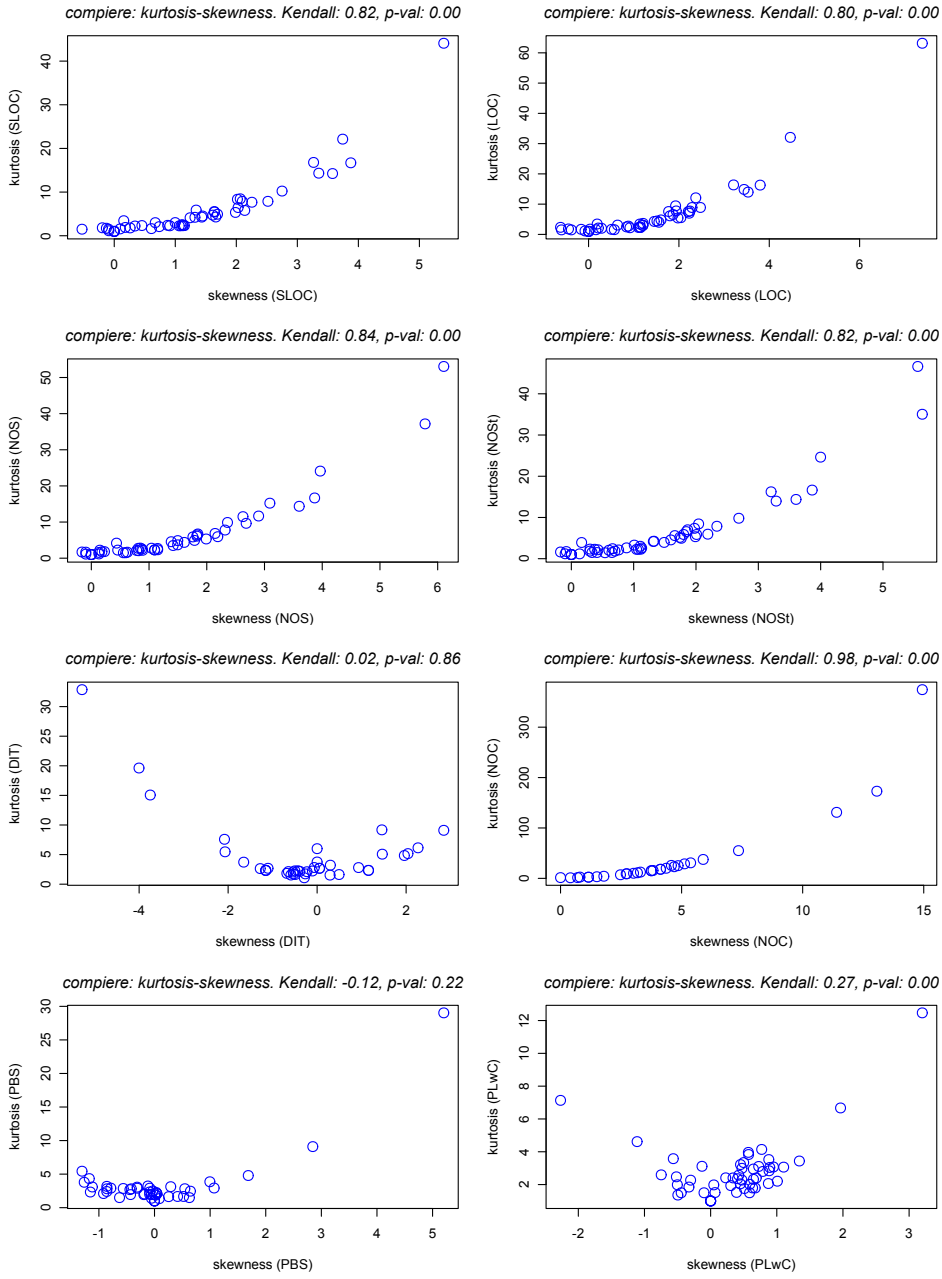

Figure 6.21: Shape of the relation between *skewness* and *kurtosis* (left), and *standard deviation* and *variance* (right).

To summarize our answer to Question 3, we note that the choice of metric does not influence neither the correlation, nor the shape of the relation between values aggregated using I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}. Apart from the perfect correlation between *standard deviation* and *variance*, also invariant with respect to the metric, correlation between values aggregated using I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson} is the highest observed among the aggregation techniques considered.

In contrast, the choice of metric does influence the correlation between *mean* and I_{Kolm}, which is consistently high and statistically significant only for the size metrics, as well as the shape of the relation between them.

6.3.3 Which index to choose?

Answering Questions 1 and 2 allows us to provide a guideline when different inequality indices should be used. It might seem that since I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson} convey the same information, all these indices are equally appropriate. However, this is not true since different indices have different application domains, emphasize different dimensions of inequality and possess different decomposability properties, as discussed in Section 3.2.

I_{Theil}, I_{MLD} and I_{Atkinson} are not applicable to negative values, while I_{Gini} and I_{Hoover} can be applied to any values as long as the mean of the values being aggregated differs from 0. We stress, however, that the range of I_{Gini} and I_{Hoover} becomes $\mathbb{R}$ in presence of negative values, challenging interpretation of the aggregated values.

Finally, if the inequality index is intended to be used to explain the inequality observed, the inequality index should be decomposable. I_{Theil} and I_{Atkinson} are the only decomposable indices of the four, hence the only ones that can be used to also explain inequality [99].

6.4 Studying the evolution of the correlation between aggregation techniques

In the second series of experiments we study the evolution of the Kendall correlation between the aggregation techniques on the Evolution Corpus, and we answer Question 4 from Section 6.1. In order to better understand the evolution of the correlation, we employ two thresholds for statistical significance: we draw the correlation coefficients supported by two-sided p-values at most equal to 0.01 as *filled blue squares*, those supported by two-sided p-values between 0.01 and 0.05 as *empty blue squares*, and finally those supported by two-sided p-values above 0.05 as *empty blue triangles*. To illustrate the evolution of the correlation between different aggregation techniques on the Evolution Corpus we choose *Hibernate* as a representative example[3].

In the previous section we have observed high and statistically significant correlation between I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}, regardless of the metric considered. This observation is consolidated by analyzing the evolution of the correlation between these inequality indices on the Evolution Corpus. For example, Figures 6.22 and 6.23 show the evolution of the correlation between I_{Gini} and I_{Theil} for the size, low-variance, and limited-range metrics considered. We observe that the correlation coefficient does not drop below 0.6 and is always statistically very significant for all metrics except DIT and NOC, for which in version 0.8.1 Hibernate was too small for the correlation to be high and statistically significant (69 files totaling 7837 SLOC).

However, the evolution is different between I_{Theil} and I_{Kolm} (Figures 6.24 and 6.25). For the size metrics we observe that the correlation became statistically significant and, although still average (0.5), the correlation coefficient significantly increased (from 0.2–0.3) starting with version 3.0. Closer inspection revealed that *Hibernate* underwent a significant increase in size when moving from v2.1.8 (29 packages) to v3.0 (109 packages). The same significant increase in size starting with version 3.0 has an effect on the correlation between I_{Theil} and I_{Kolm} for DIT, NOC, and PLwC as well. For NOC, correlation between I_{Theil} and I_{Kolm} switched from positive to negative, but remained very low and statistically insignificant. For DIT and PLwC, correlation between I_{Theil} and I_{Kolm} started increasing and became statistically significant.

[3]The other systems in the Evolution Corpus exhibit similar patterns. For complete results see `www.student.tue.nl/X/b.n.vasilescu/evolution/evolution.html`

Evolution of correlation between I_{Gini} and I_{Theil}

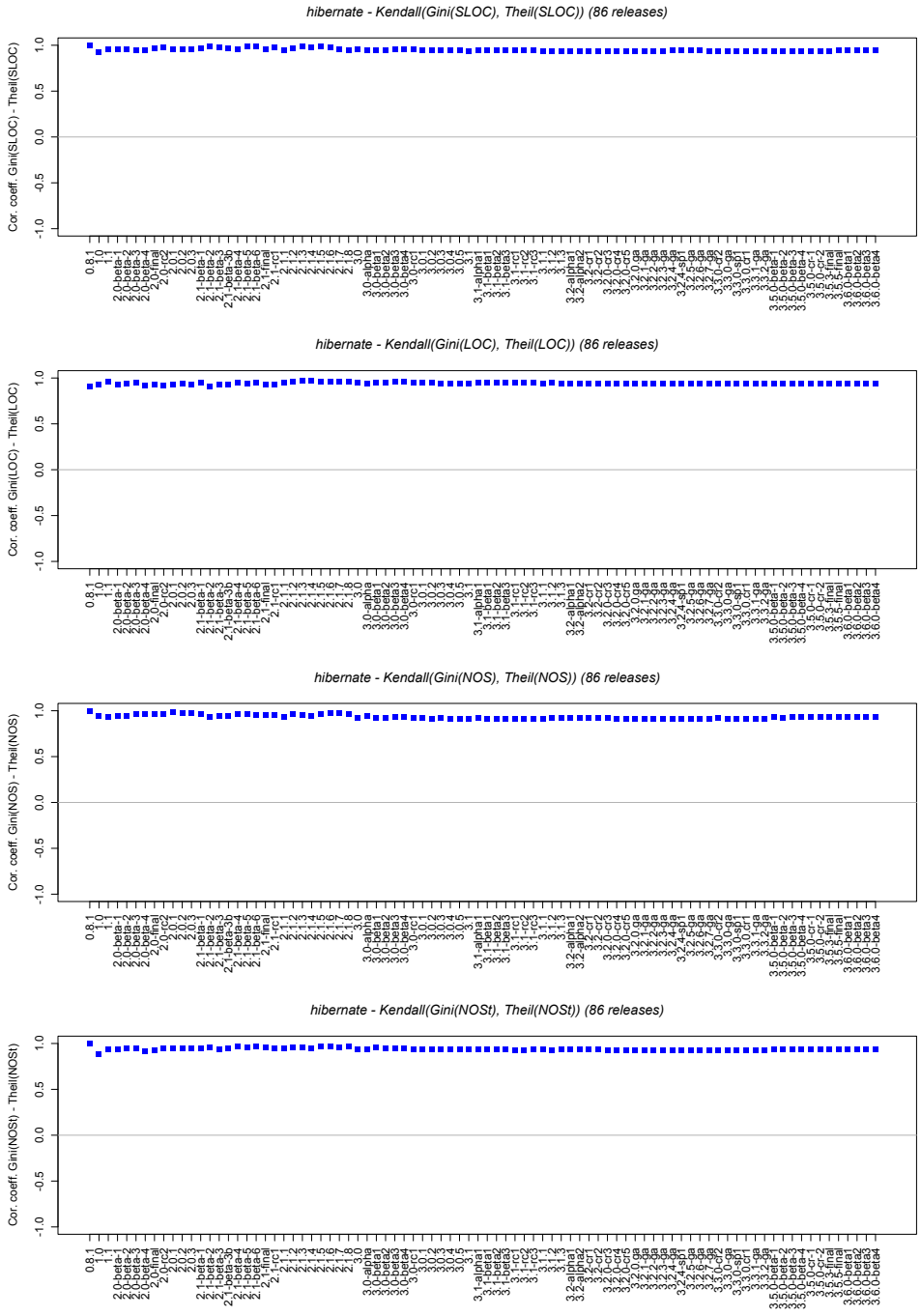

Figure 6.22: Correlation between I_{Theil} and I_{Gini} is almost perfect for the size metrics.

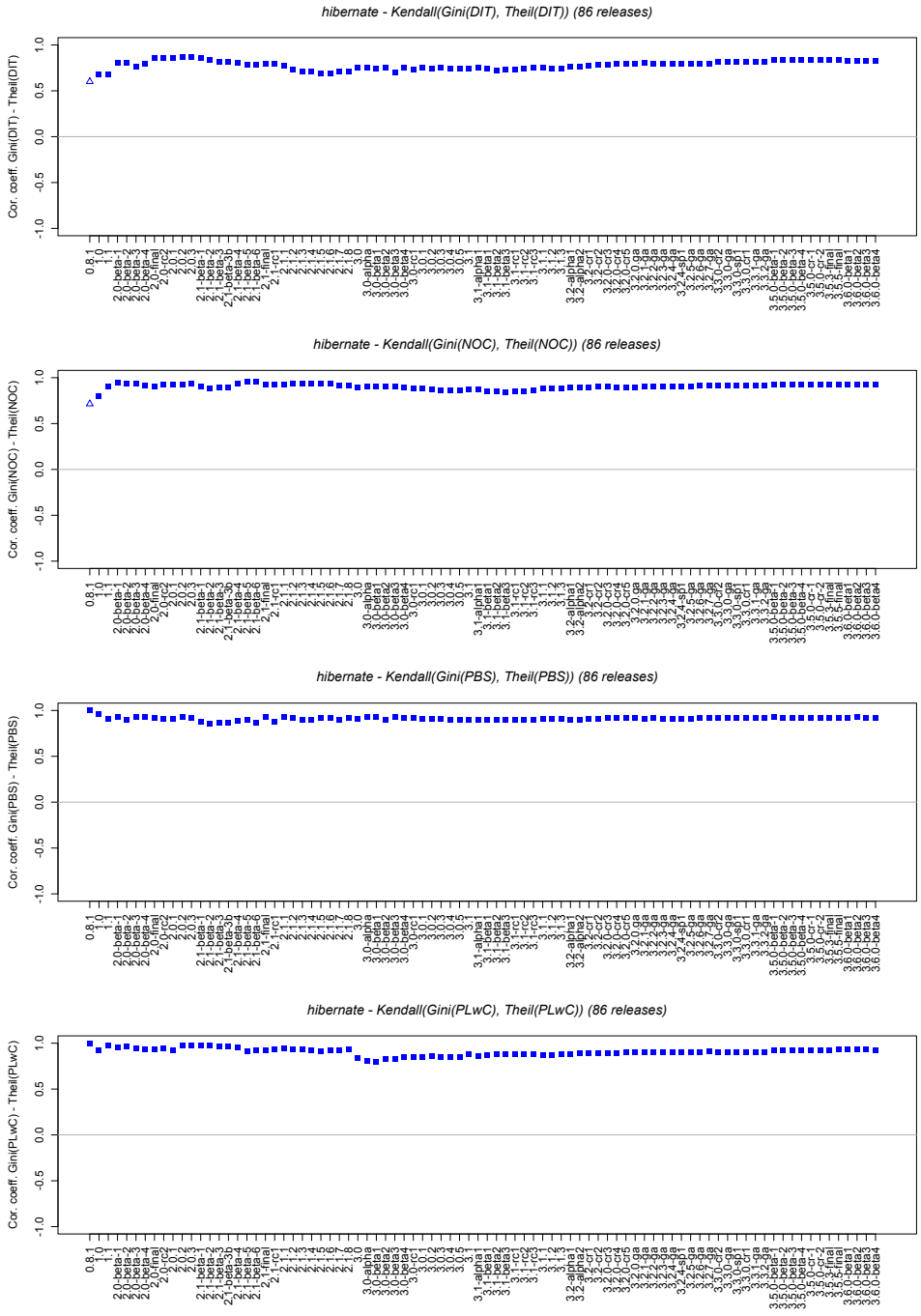

Figure 6.23: Correlation between I_{Theil} and I_{Gini} is always high for the low-variance metrics and the limited-range metrics.

Evolution of correlation between I_{Kolm} and I_{Theil}

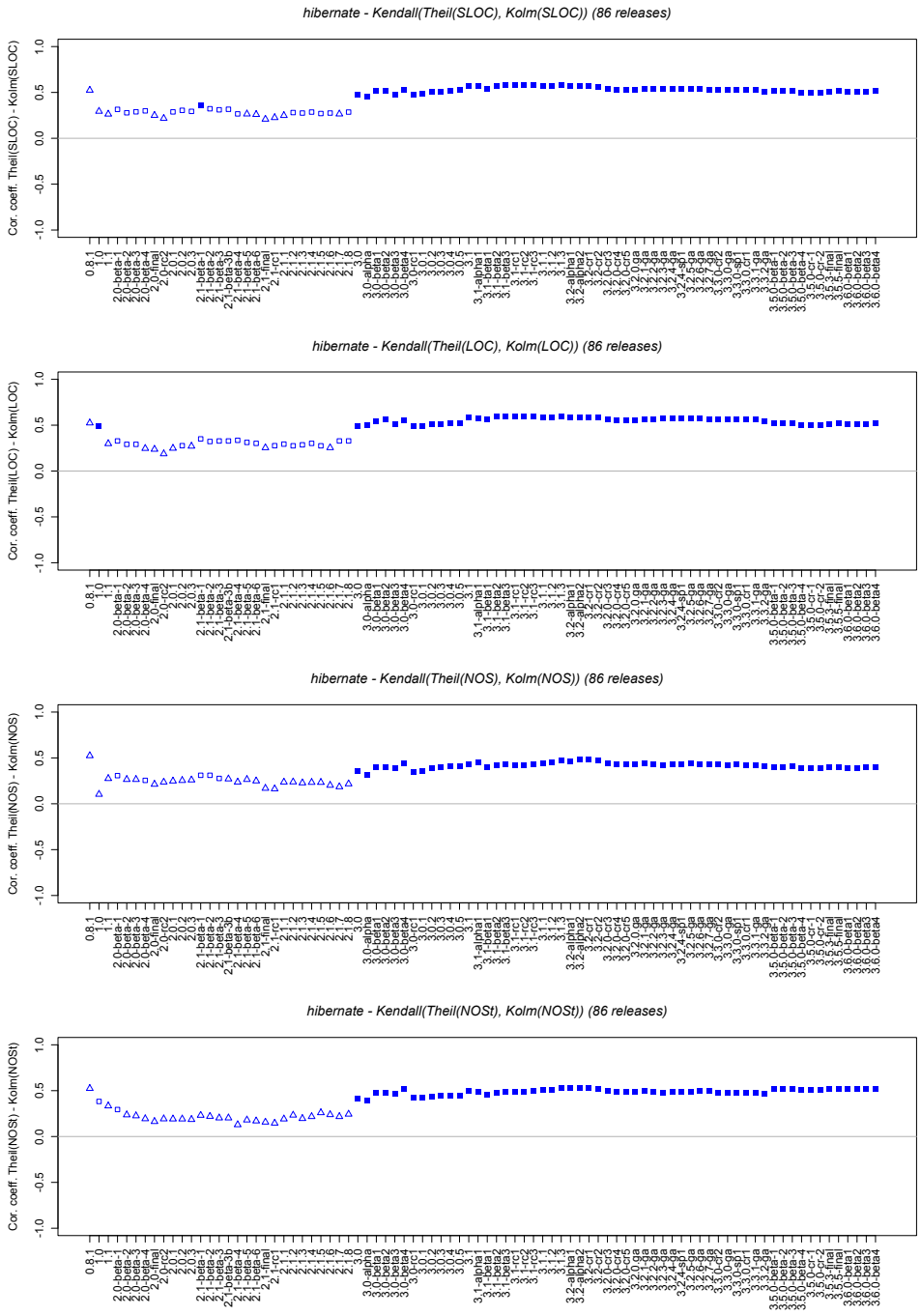

Figure 6.24: Correlation between I_{Theil} and I_{Kolm} for the size metrics became statistically significant after Hibernate increased in size.

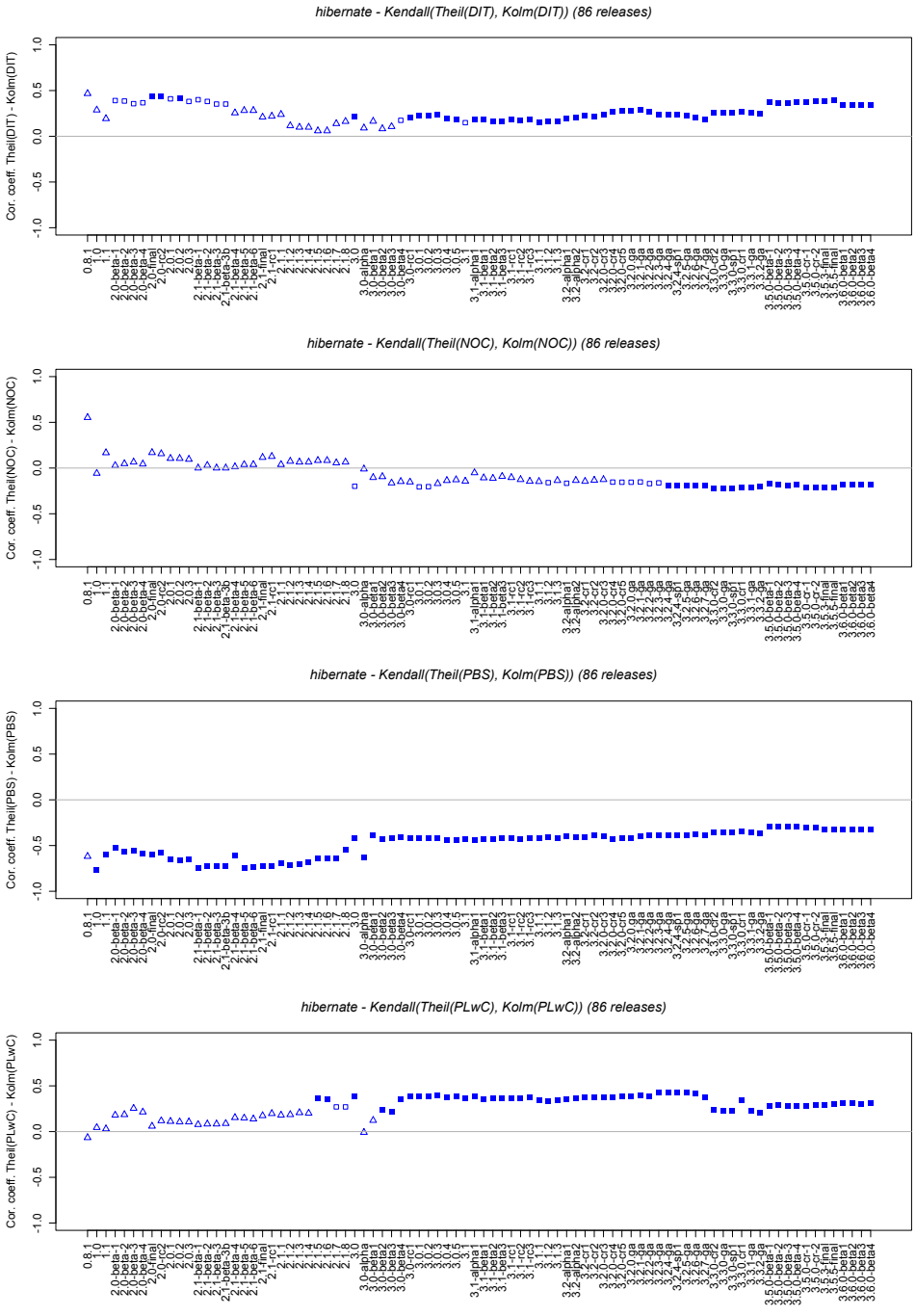

Figure 6.25: Correlation between I_{Theil} and I_{Kolm} is inconsistent for the low-variance metrics and the limited-range metrics.

An interesting case is the correlation between *mean* and I_{Kolm}, which we previously observed to be high (0.8) and statistically significant in 90% of the Corpus for the size metrics. On the other hand, in the Evolution Corpus, there are significant variations in the correlation coefficient between different versions of the system. We illustrate this in Figure 6.26 on *Hibernate* and *Weka* for SLOC. While correlation between *mean* and I_{Kolm} seems to fluctuate without clear relation to system size, "jumps" of the correlation values in both graphs of Figure 6.26 show major releases such as 3.5.0 in *Hibernate* and 3.6.0 in *Weka*.

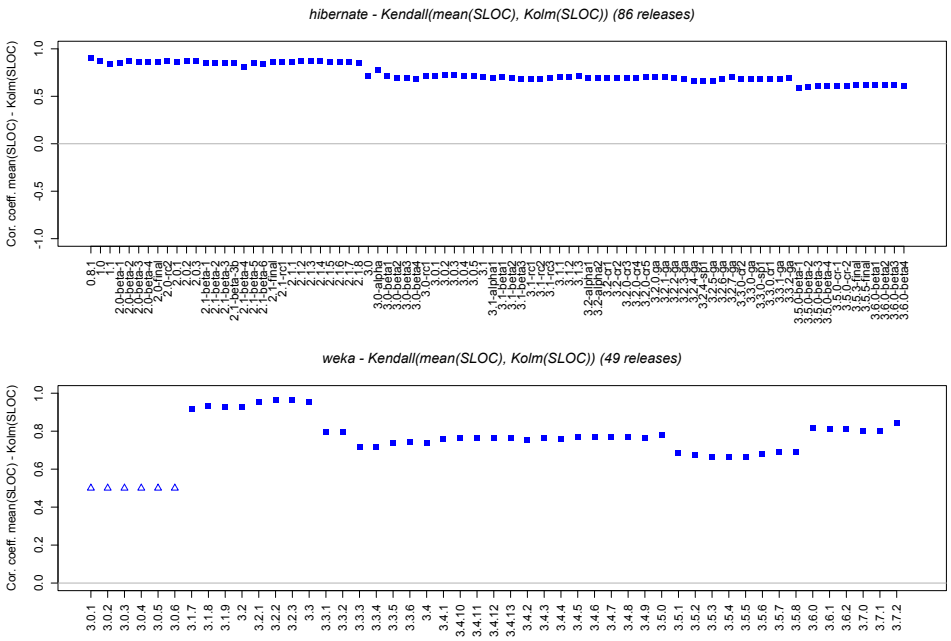

Figure 6.26: Correlation between *mean* and I_{Kolm} fluctuates without clear relation to system size.

To summarize our answer to Question 4, we note that:

- Consistently high (> 0.8) and statistically significant correlation can be observed between I_{Gini}, I_{Theil}, I_{Hoover}, I_{MLD} and I_{Atkinson} across the Evolution Corpus for all metrics.
- Correlation between I_{Theil} and I_{Kolm} increases as the system size increases for the size metrics, but has inconsistent behaviour for the other metrics.
- Correlation between *mean* and I_{Kolm} fluctuates without clear relation to system size.

6.5 Threats to validity

Although this correlation study addresses some of the threats to the validity of the pilot studies on which it it founded (e.g., multiple metrics, representative set of benchmarks, representative set of versions), a number of threats to validity should still be addressed in the future.

First, while the results for the size metrics agreed in all cases, we have observed inconsistent results between the low-variance metrics DIT and NOC, and between the limited-range metrics PBS and PLwC. Therefore, we intend to furthermore extend the study by considering other software metrics.

Second, we have used default parameters for I_{Atkinson} and I_{Kolm}, as discussed in Section 3.1. However, the choice of parameters influences the sensitivity of the two indices to transfers between the "rich" and the "poor", thus potentially influencing the strength and the nature of the relation with other aggregation techniques. Therefore, a more extensive evaluation using different parameter values for I_{Atkinson} and I_{Kolm} is desirable.

Third, we have used external tools to compute the chosen metrics, i.e., Understand[4] and SourceMonitor[5], for which we can neither guarantee the correctness of the metrics' definitions, nor the quality of the results. For example, we have discovered incorrect values for the size metrics computed using Understand in certain situations, and we have adjusted the results accordingly. Therefore, an extensive validation of the metrics data is desirable.

6.6 Conclusions

This study represents an extension to the pilot studies in Chapter 5 along three directions.

First, we have extended the pilot studies to a number of size metrics, i.e., *number of source lines of code* (SLOC), *number of lines of code* (LOC), *number of semicolons* (NOS), and *number of statements* (NOSt), limited-range metrics, i.e., *percentage of branching statements* (PBS) and the *percentage of lines with comments* (PLwC) [82], and low variance metrics, i.e., *depth of inheritance tree* (DIT) and the *number of children* (NOC) [22,108]. Second, we have extended the pilot studies to a representative set of benchmarks, i.e., the 106 open-source Java systems comprising the Qualitas Corpus. Third, we have considered a representative set of versions for 13 systems (out of the 106 systems in the Qualitas Corpus) with 10 or more versions available, in the form of the Evolution Corpus.

As a result of the pilot studies we have observed (i) high correlation between I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}, and (ii) similarly between

[4] `http://www.scitools.com/`

[5] `http://www.campwoodsw.com/sourcemonitor.html`

I_{Kolm}, *mean*, *standard deviation*, and *variance*, suggesting that values aggregated using these techniques convey the same information. Moreover, we have observed (iii) high correlation between values aggregated using *skewness* and *kurtosis*.

The current study confirms (i), since I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover} and I_{Atkinson} show consistently high and statistically significant correlation between them, i.e., the aggregates obtained using these techniques convey the same information, regardless of the metric considered. In contrast, (ii) is only partly confirmed, since the correlation between *mean* and I_{Kolm} is high and statistically significant only for the size metrics, while the correlation between I_{Kolm} and *standard deviation* or *variance* is lower and less statistically significant for all metrics. Finally, the current study partly confirms (iii), i.e., values aggregated using *skewness* and *kurtosis* convey the same information for the size metrics.

Moreover, we have analyzed the influence of the specific version of a system on the correlation between the various techniques by studying it in an evolutionary setting, on the Evolution Corpus. We have observed consistently high (> 0.8) and statistically significant correlation between I_{Gini}, I_{Theil}, I_{Hoover}, I_{MLD} and I_{Atkinson}, suggesting that values aggregated using these techniques convey the same information regardless of the metric used or version analyzed. Moreover, we have observed that correlation between I_{Theil} and I_{Kolm} increases as the system size increases for the size metrics, but has inconsistent behaviour for the other metrics. Finally, we have observed that correlation between *mean* and I_{Kolm} is inconsistent, and it fluctuates without clear relation to system size.

Chapter 7

Threshold-based aggregation techniques study

In both the pilot study in Chapter 5, as well as the extensive correlation study in Chapter 6 we have aggregated metrics data from class to package level and we have empirically evaluated the traditional and econometric aggregation techniques discussed in Chapters 2 and 3, respectively.

In this chapter we extend the pilot study by aggregating metrics from method to class level rather than from class to package level, thus enabling the comparison with the threshold-based aggregation techniques discussed in Chapter 4. For the purpose of illustration, we choose *source lines of code* (SLOC) as metric. Note that both SIG and Squale can be applied to aggregation of SLOC values measured per method since translation functions (from metrics values to individual marks) and thresholds for SLOC per method have been published in [5, 6, 76].

Specifically, we address the following questions:

1. To what extent do the SIG and Squale approaches *agree*, i.e., to what extent do they rank distributions of metrics values similarly?

2. Which and to what extent do other aggregation techniques (traditional or econometric) agree with the SIG or Squale approaches?

3. How does correlation between aggregation techniques change after changing the aggregation level from class–package to method–class?

7.1 Methodology

In order to answer the previous research questions we aggregate *source lines of code* (SLOC) values from method to class level and we investigate statistical correlation between pairs of aggregation techniques.

As case studies we chose a representative subset of 20 open-source Java systems from the Qualitas Corpus version 20101126r[1], uniformly selected such that all sizes are represented. Our subset ranges from Azureus, the fifth-largest system in the Corpus (453433 SLOC), to FitJava, the second-smallest system in the Corpus (2240 SLOC). The complete list of the systems considered is given in Table 7.1. We chose to perform the evaluation on a

Table 7.1: Subset of the Qualitas Corpus considered in the study of threshold-based aggregation techniques

Ant	AspectJ	Cayenne	Columba	DisplayTag
ANTLR	Axion	CheckStyle	Compiere	DrawSWF
Aoi	Azureus	Cobertura	C_JDBC	DrJava
ArgoUML	Castor	Colt	Derby	FitJava

subset rather than the entire Corpus since method-level metrics data as computed by Understand[2] required manual postprocessing in order to be accurately extracted from the result files. Specifically, classes with a name starting with a lowercase letter would be ignored by our tooling, as well as all the methods defined therein, unless classes would be (manually) renamed to start with an uppercase letter. This data sanitization process does not affect the validity of our results since it improves the quality of the data.

As with the pilot studies, we stress that statistical correlations are not transitive [62], i.e., we have to consider *all pairs* of aggregation techniques. As with the study in Chapter 6, we focus on source code of the actual systems and we exclude libraries. Similarly, we use the Corpus metadata to discern between what is identified as actual source code of a system, and what is considered third-party.

For all pairs of aggregation techniques compared, we considered classes containing at least 2 methods. This is motivated by the fact that, when applied to classes containing only one method, most of the traditional aggregation techniques (standard deviation, variance, skewness, and kurtosis) are undefined, and all inequality indices are equal to 0. For each class in each system, we aggregate the SLOC values of all the methods defined in that class, in turn, using each of the aggregation techniques considered. We implemented the SIG approach using the translation function from metrics to individual marks described in [76] for the three weights typically used in practice [76], i.e., 3, 9 and 30. For the SIG approach, we implemented the discrete version, in which the rating is an integer number, as well as the continuous version described in [5], in which the rating is a real number, both in the range $[1, 5]$, with the thresholds in $[5, 6]$.

[1]Qualitas Research Group, Qualitas Corpus Version 20101126, The University of Auckland, February 2009. `http://qualitascorpus.com`.

[2]`http://www.scitools.com/`

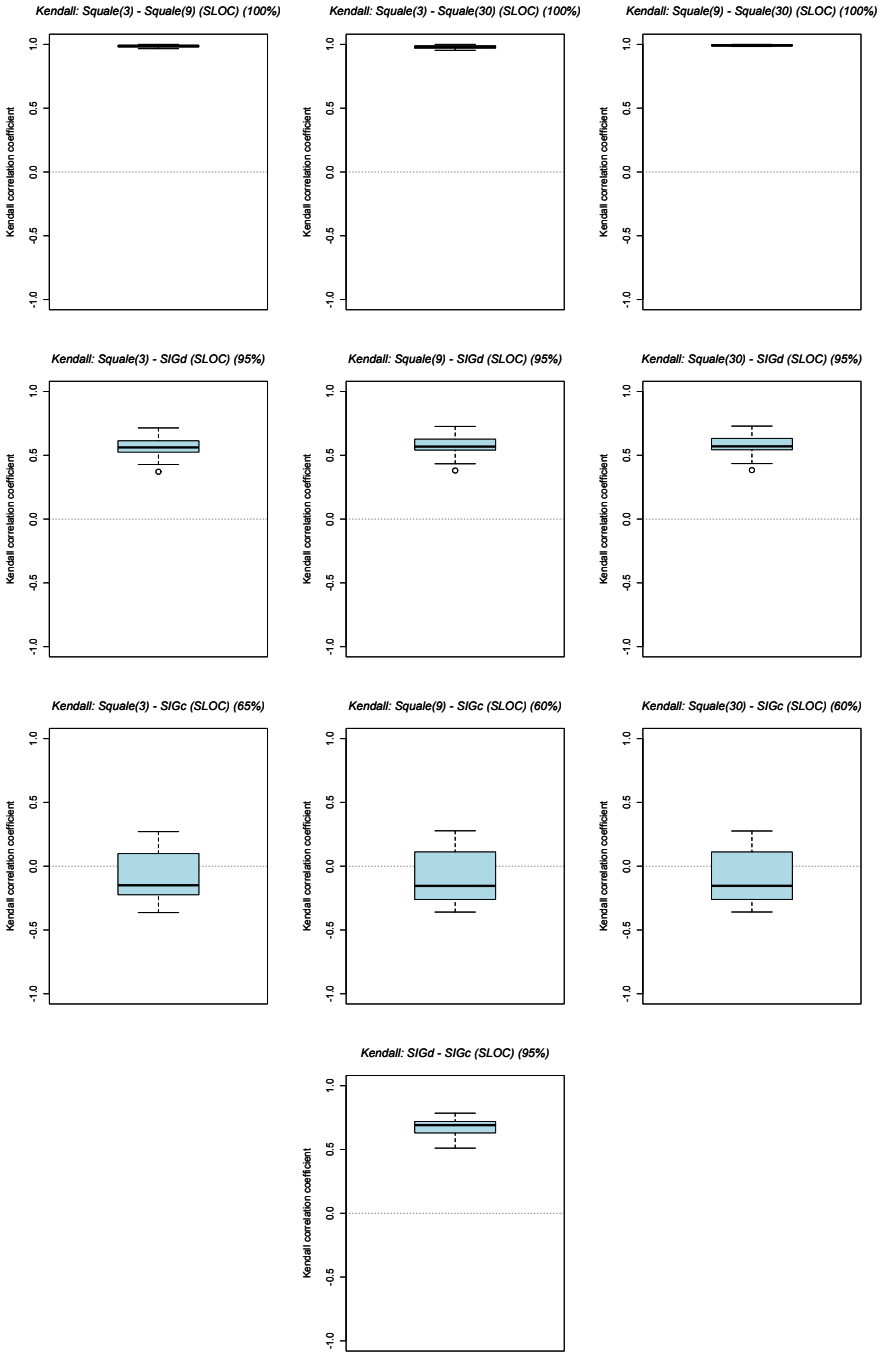

Figure 7.1: Squale correlates with discrete SIG. Discrete SIG correlates with continuous SIG. However, Squale does not correlate with continuous SIG.

As discussed in the data analysis methodology of the pilot and extensive correlation studies, we opt for Kendall's rank correlation coefficient τ [57] to measure statistical correlation between values aggregated using different techniques. All computations were performed using R [92].

7.2 Results

7.2.1 Do the SIG and Squale approaches agree?

We start by studying the correlation between the threshold-based aggregation techniques (Figure 7.1), and answer Question 1.

Note that the exact weights used for Squale, as well as the percentage of the systems for which the correlation is statistically significant are displayed in between parentheses in the boxplots.

We observe almost perfect correlation between values aggregated using Squale with the three different weights, statistically significant for all systems (Figure 7.1 first row), i.e., values aggregated with Squale using different weights convey the same information.

Moreover, we observe above average correlation (0.5–0.6) between values aggregated using Squale and the discrete version of SIG (Figure 7.1 second row), statistically significant for 95% of the systems. Similarly, we observe above average correlation (0.6–0.7) between values aggregated using the discrete and continuous versions of SIG (Figure 7.1 last row), again statistically significant for 95% of the systems. However, correlation between Squale and the continuous version of SIG (Figure 7.1 third row) is another good example of the non-transitivity of statistical correlations, since it is very low and much less statistically significant.

7.2.2 Do other techniques agree with SIG or Squale?

Figures 7.2–7.4 display the results for Squale with weight 3, and the discrete and continuous versions of SIG.

In order to answer Question 2, we start by observing that Squale (Figure 7.2) shows average negative correlation with the inequality indices and *mean*, *sum*, *standard deviation*, and *variance*, statistically significant for 95% of the systems. The correlation between Squale and other aggregation techniques, albeit statistically significant for at least 90% of the systems, is much lower.

In contrast, the *discrete* version of SIG (Figure 7.3) shows very low negative correlation with the other aggregation techniques, again statistically significant for at least 90% of the systems. Finally, the *continuous* version of SIG (Figure 7.4) does not correlate with any of the other aggregation techniques.

Squale

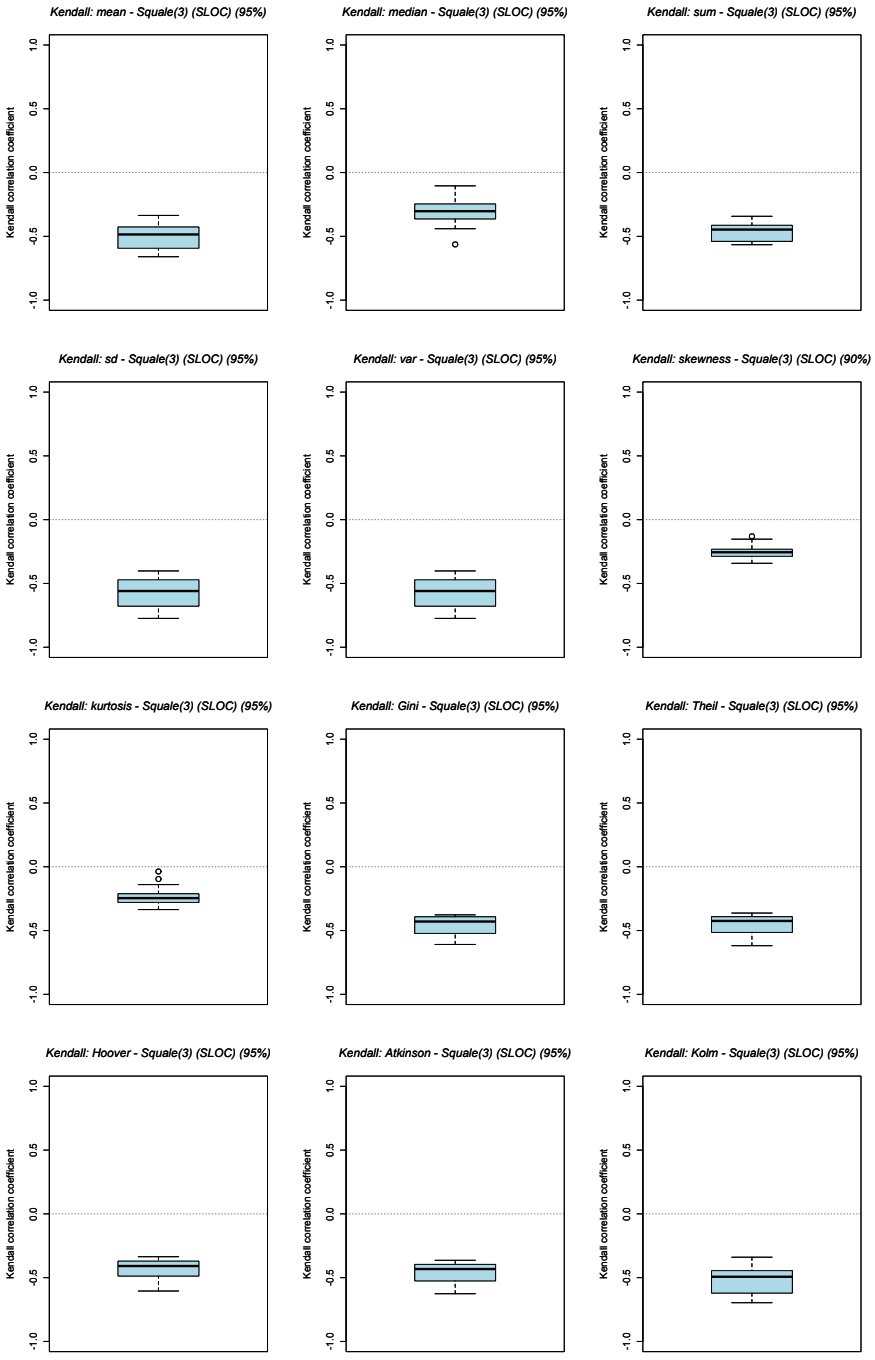

Figure 7.2: Squale shows average negative correlation with the inequality indices and *mean*, *sum*, *standard deviation*, and *variance*.

Discrete SIG

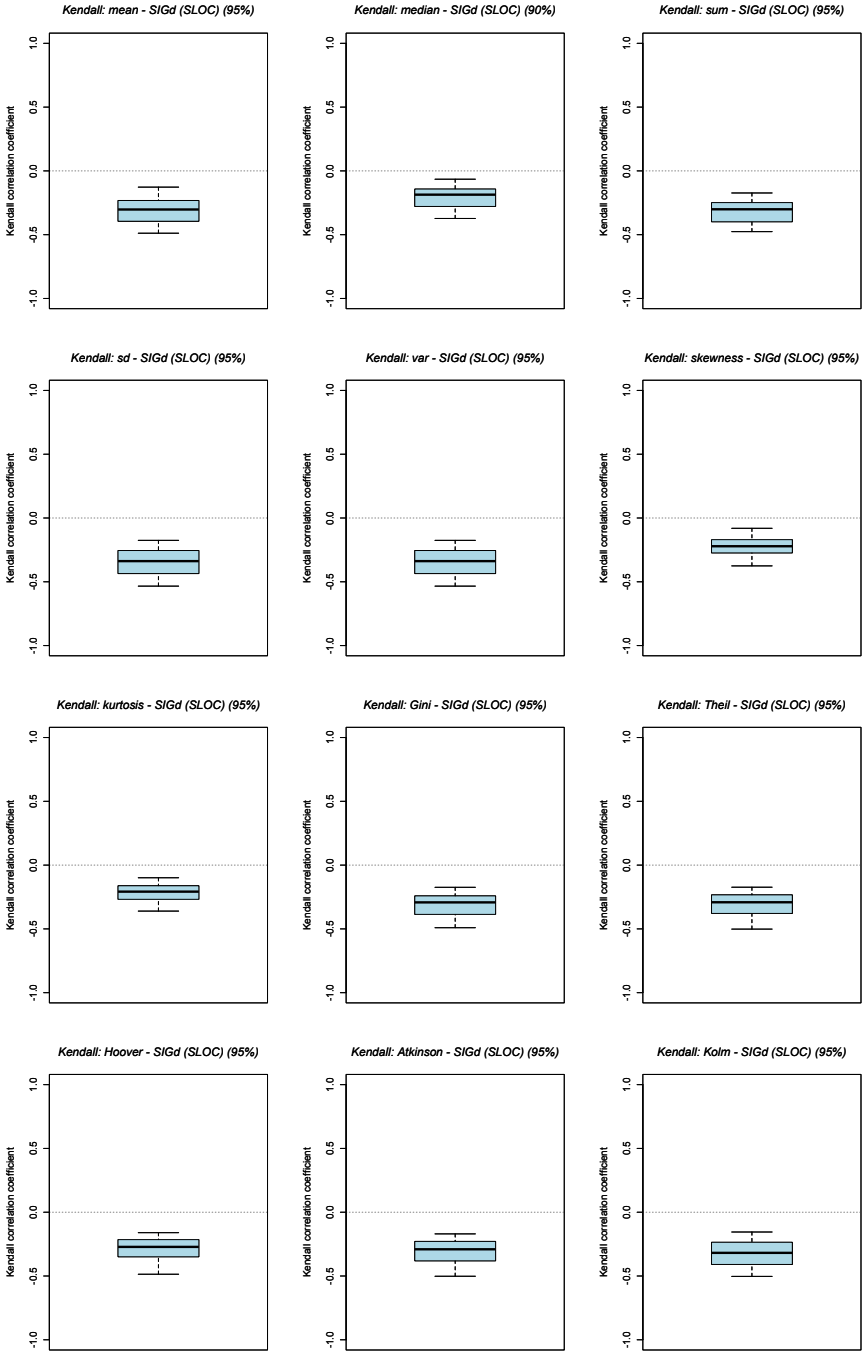

Figure 7.3: The *discrete* version of SIG shows very low negative correlation with the other aggregation techniques.

Continuous SIG

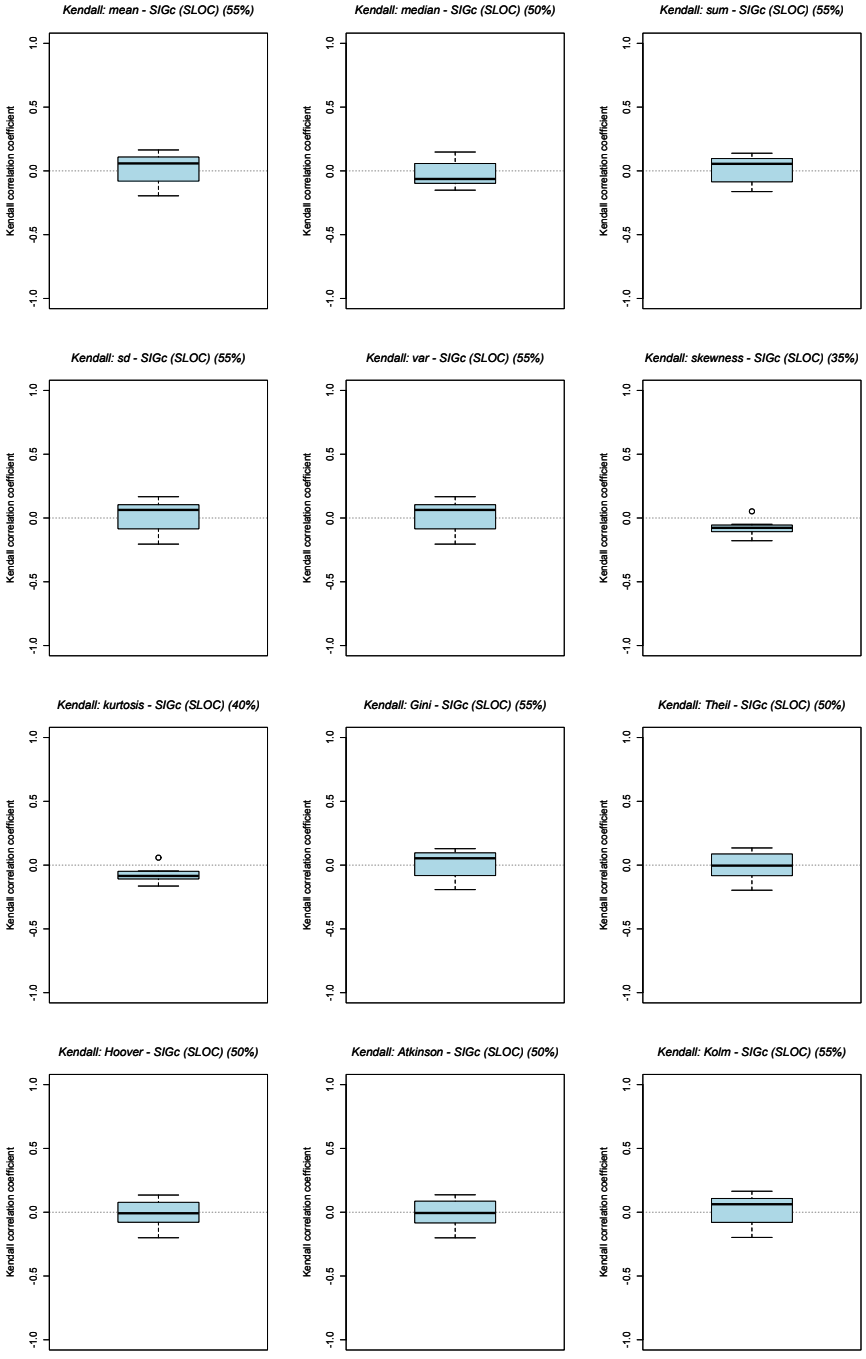

Figure 7.4: The *continuous* version of SIG does not correlate with any of the other aggregation techniques.

7.2.3 Does the aggregation level influence correlation?

In order to answer Question 3 we refer to Figure 7.5, depicting the correlation coefficients for a number of pairs of inequality indices for which we previously observed high and statistically significant correlation when aggregating from class to package level.

Similarly, we now observe very high and statistically significant correlation between values aggregated using I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}, while the correlation between I_{Kolm} and the other inequality indices, albeit also statistically significant, is significantly lower and more spread out.

7.3 Threats to validity

In addition to the threats to validity discussed in Sections 5.4 and 6.5, we identify the following main threats to the validity of results presented in this study. First, we have only considered a subset of the Corpus, and only a single version from each system. Therefore, we need to consider a more representative set of benchmarks, and a more representative set of their versions.

Second, we have presented results for a single metric, i.e., number of source lines of code. Subsequent studies should investigate whether the results obtained can be generalized to additional metrics.

7.4 Conclusions

We presented the results of an extension to the pilot studies in Chapter 5, in which we aggregated SLOC values from method to class level on a subset of the Qualitas Corpus, and studied statistical correlation between all aggregation techniques discussed in Chapters 2, 3, and 4.

We observed that the Squale and the discrete version of the SIG threshold-based aggregation techniques show negative average correlation with other aggregation techniques, and positive average correlation among themselves, i.e., there is redundancy in the information conveyed by values aggregated using Squale or discrete SIG, but both measure different characteristics than any of the other aggregation techniques.

In addition, we observed that changing the aggregation level with respect to the studies in Chapters 5 and 6 did not affect the correlation between the inequality indices, as measured in Chapter 6.

Influence of aggregation level

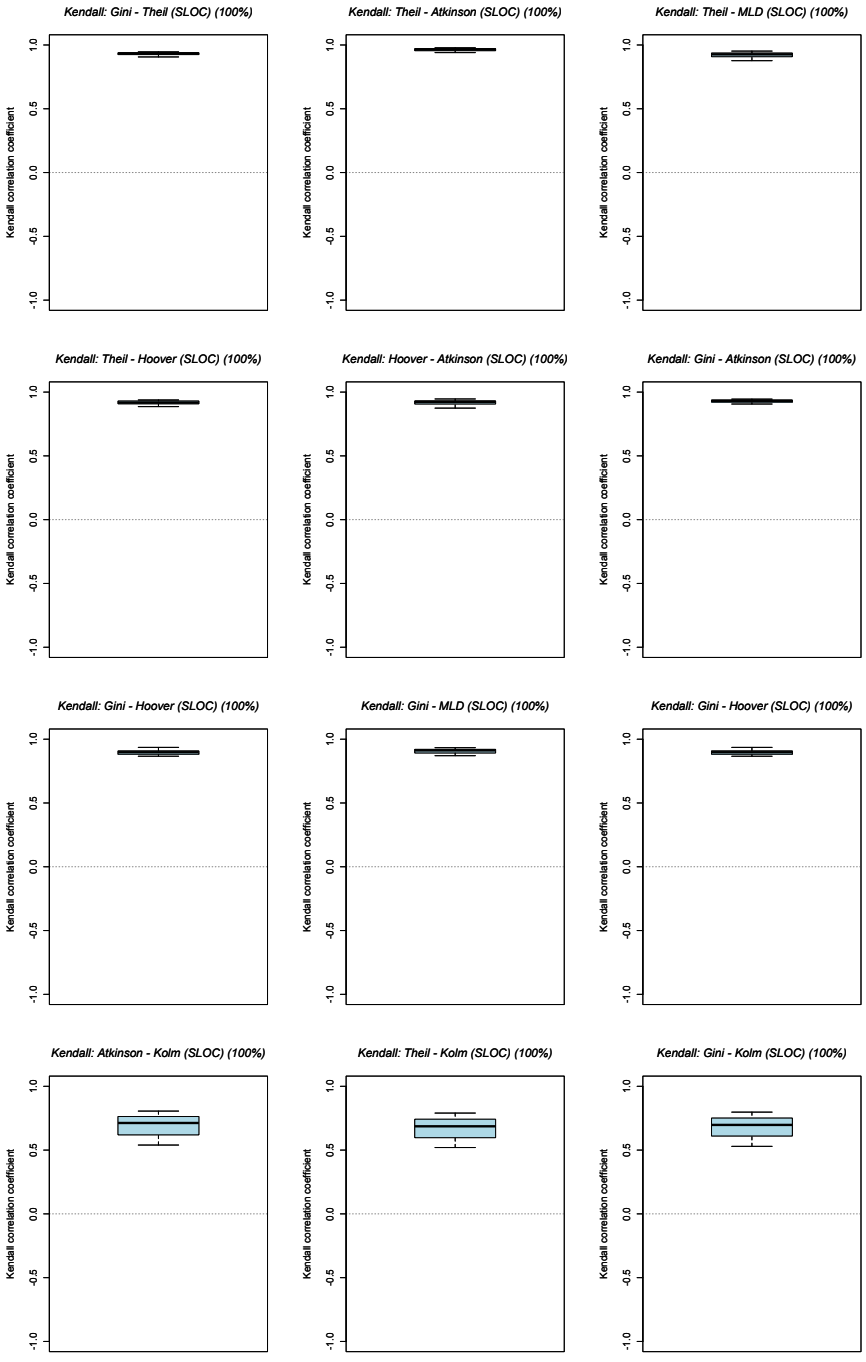

Figure 7.5: Aggregating from method to class level does not significantly affect correlation between the inequality indices as measured in Chapter 6.

Chapter 8

Highlighting undesirable values in the aggregate

The aggregation in Squale, as part of a software quality model, was designed to highlight undesirable metrics values (and undesirable, i.e., low, individual marks) in order to warn the software engineers in case of potential problems. In contrast, it is not the goal of inequality indices to highlight lower values (poorer people) over higher ones (richer people). However, different inequality indices are known to have different sensitivities to either poorer or richer people, as discussed in Section 3.2.

However, I_{Squale} and the inequality indices have never been empirically compared before using software metrics data. In this section we compare the reaction of I_{Squale} to that of the inequality indices in the presence of an increasing large number of low individual marks (IMs). Recall that the aggregation in Squale is a two-step process, requiring that the values of metrics are first translated to IMs, which are then aggregated to a global mark using the approach in equation 4.3. Therefore, in order to enable the comparison, we apply the inequality indices to aggregation of IMs rather than raw metrics data as well.

8.1 Methodology

In our experiment we consider different variants of the same software system inspired by Eclipse 2.0, and we aggregate the individual marks computed for source lines of code measured per method. We consider I_{Squale} with weights 3, 9, and 30, I_{Theil}, I_{MLD}, I_{Gini}, I_{Kolm}, I_{Atkinson}, and I_{Hoover} (see Sections 4.2 and 3.1 for definitions, respectively).

Version 2.0 of Eclipse, selected for convenience reasons, has a total of 8612 methods, from which 8093 have an IM of 3 (perfect mark), and 519 have IMs lower than 3 according to the translation function in Figure 4.2 (i.e., SLOC > 36). Therefore, as base ("perfect") case for the experiment

we consider that Eclipse 2.0 has only perfect IMs (i.e., 8612 methods with SLOC ≤ 36). We then gradually "pollute" the data by replacing a number of perfect IMs by imperfect IMs, selected from the 519 available imperfect IMs extracted from Eclipse. For example, for the test with 10% imperfect IMs, we have a random selection of 7751 perfect IMs and 861 imperfect ones. When we need more imperfect IMs than Eclipse actually contains, we select the same ones several times.

At each step we compute the aggregated value according to all the aggregation techniques considered (i.e., the inequality indices and I_{Squale}).

Therefore, we control two variables independently:

- Quantity of imperfect IMs: we perform experiments with the following quantities of imperfect IMs: 0% (i.e., base case); 10%, 20%, ..., 100%
- Value of the imperfect IMs: we perform different experiments with IMs chosen successively in the intervals: $[2, 3)$, $[1, 2)$, $[0.5, 1)$, $[0.1, 0.5)$, and $[0, 0.1)$.

Because we select the imperfect IMs randomly, we repeat each experiment 10 times and present the mean of the aggregated result.

8.2 Results

Figure 8.1 presents the results for the arithmetic *mean*. We observe that even with 30% very bad marks (IMs in the range $[0, 0.1)$), the aggregated result is still greater than 2, which would still indicate good quality according to the Squale rating scale.

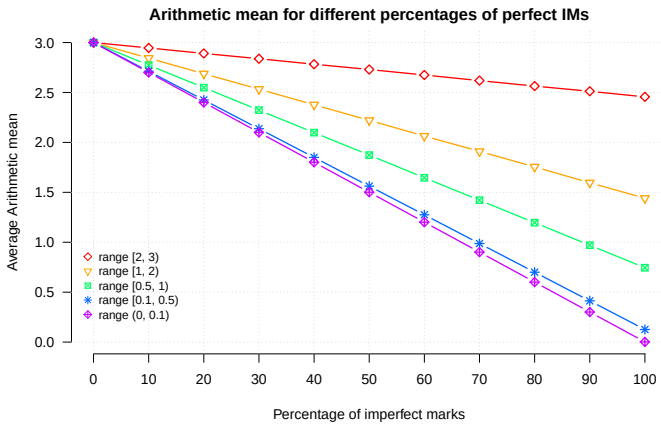

Figure 8.1: Results of experiments for arithmetic mean

The results of this first graph are repeated in all other graphs in the form of a grey triangle in the background to ease comparison of all aggregation techniques to the upper and lower bounds of the results for arithmetic *mean*.

The results for I_{Squale} (Figure 8.2) show that it behaves as expected, with the gradation of the different weighting (from soft $\lambda = 3$ to hard $\lambda = 30$). Particularly, the hard weighting results in a low aggregated result even for a small quantity (10%) of bad IMs.

For the econometric indices (Figures 8.3 and 8.4), one should recall that they do not discern between all values being equal but high, and all values being equal but low. Therefore, all inequality indices are zero when the aggregated values are all equal (e.g., base case). Since this characteristic is opposite to that of I_{Squale}, which results in a perfect result (i.e., 3) when all IMs are perfect (i.e., 3), to ease comparison with I_{Squale} we inverted the y-axis of the results for the inequality indices (i.e., on the left of the graphs; the right y-axis is for the grey triangle referring to the arithmetic mean).

Moreover, for the same reason, we observe for all inequality indices that inequality increases as long as the perfect IMs form the "majority", and then inequality starts decreasing (curve going up on our inverted axis) when the imperfect IMs become the norm rather than the exception. However, the inflexion point differs between the six inequality indices. For example, for imperfect IMs selected in the range $[0.1, 0.5)$, inequality measured using I_{Gini} starts decreasing when 90% of the IMs are imperfect, while inequality measured using I_{Hoover} starts decreasing when 80% of the IMs are imperfect, and inequality measured using I_{Kolm} starts decreasing when only 30% of the IMs are imperfect.

In comparison to I_{Squale}, I_{Kolm} (Figure 8.4) is the inequality index that best highlights bad results (i.e., low IMs), as long as there are not too many of them (up to 30% or 40%, depending on the range of the imperfect IMs).

Although I_{Kolm} decreases when the proportion of bad results augments beyond this threshold (since then there is less inequality), in practice I_{Kolm} could still be used to highlight bad results, as one can hope that the majority of components will have acceptable results (e.g., because software metrics are usually positively skewed [14,46]). In contrast, our experiments suggest that I_{Theil} (Figure 8.3) is the inequality index that least highlights bad results (i.e., low IMs), even less than the arithmetic mean.

More worrying for I_{Kolm} is the fact that an improvement of the quality (e.g., from 60% to 50% imperfect IMs) will result in an augmentation of inequality (from a majority of imperfect methods to less) and, therefore, a worsening of the aggregated value.

Note that although it may appear in Figures 8.3 and 8.4 that, e.g., I_{Theil}, I_{MLD}, and I_{Atkinson} are constant when the range of imperfect marks is $[2, 3)$ for all percentages of imperfect marks, this fact is due to the relatively small variations in these indices compared to the range $(0, 0.1)$ of imperfect marks.

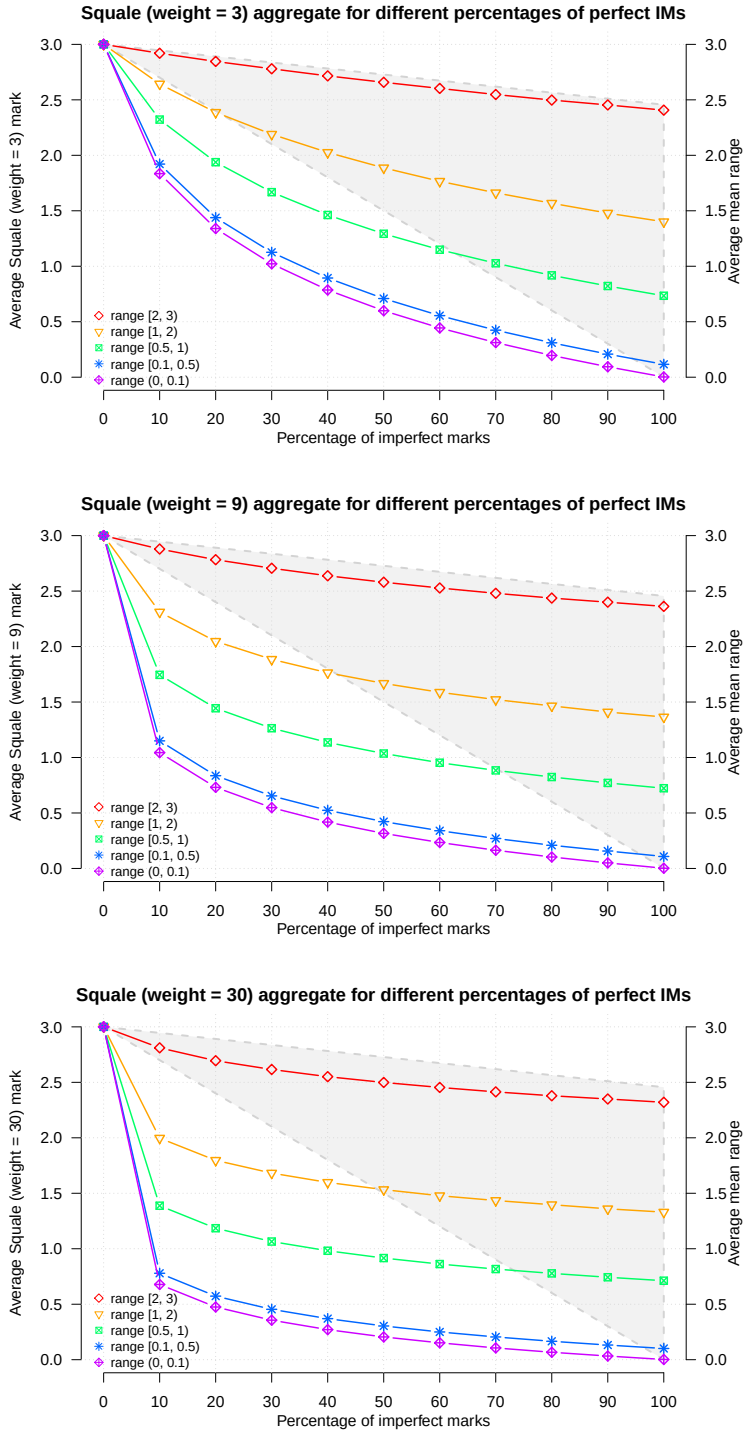

Figure 8.2: Results of experiments for I_{Squale} with different weights.

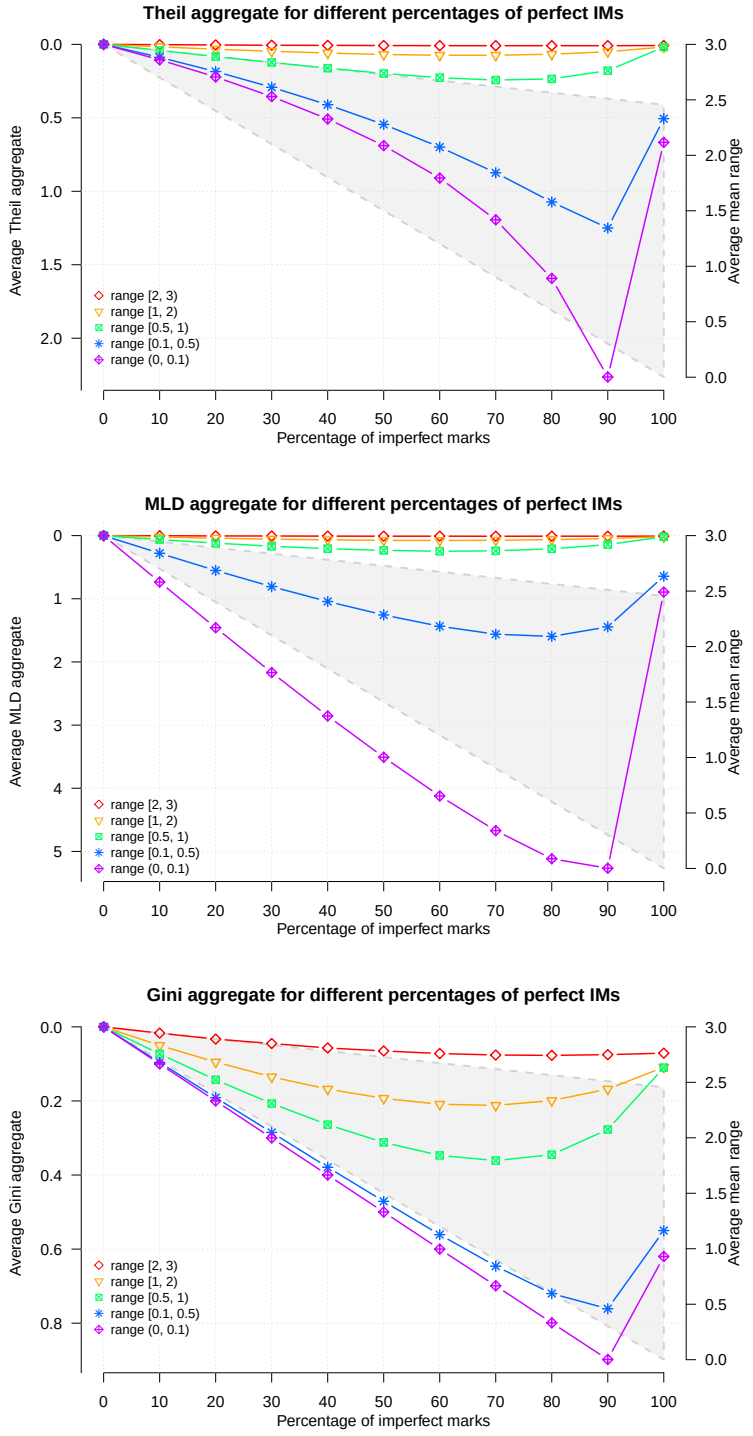

Figure 8.3: Results of experiments for I_{Theil}, I_{MLD}, and I_{Gini}.

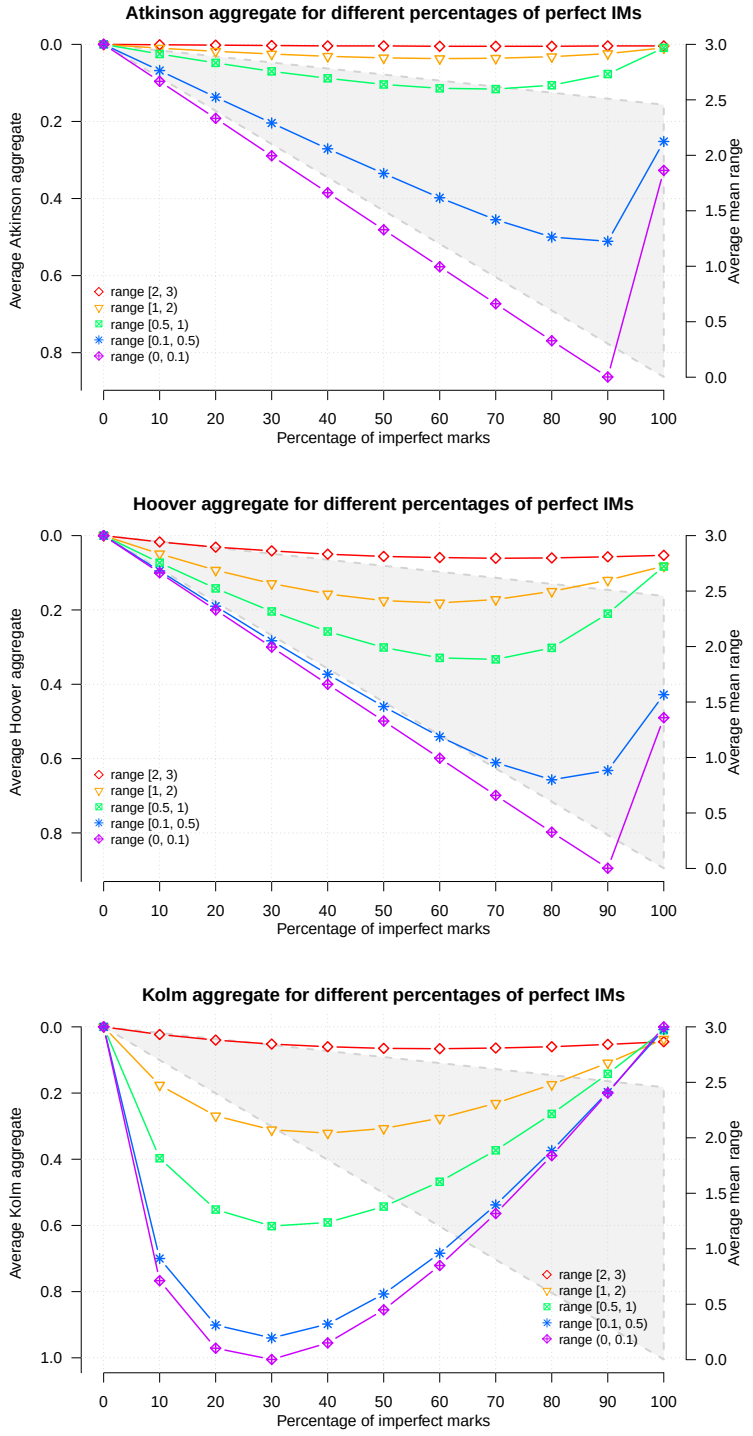

Figure 8.4: Results of experiments for I_{Atkinson}, I_{Hoover}, and I_{Kolm}.

8.3 Threats to validity

The results presented above a subject to a number of threats to validity, which should be addressed in the future.

With respect to data selection, in each experiment we are considering artificial data with only a limited range of imperfect IMs (e.g., $[0.5, 1)$), whereas in real projects the imperfect IMs would have more spread. This aspect could potentially impact the results of the inequality indexes, which would probably decrease since the increase in spread would imply a decrease in inequality.

Moreover, there are only a limited total number of imperfect IMs (519), hence an even more limited number of imperfect IMs in a certain range. Since when we need more imperfect IMs than actually available we select the same ones several times, inequality of IMs in our experiment may be lower than that of IMs in real projects.

Furthermore, our separation of undesirable values into classes assumes the thresholds 0.1, 0.5, 1, and 2 for the five ranges considered, and it is not clear whether the results can be generalized to different separation criteria, such as the separation into classes according to the actual values but using different thresholds, and the separation into classes according to percentiles rather than according to the actual values.

In addition, the aggregation is performed on IMs (i.e., SLOC values normalized into the range $[0, 3]$) rather than raw metrics data, which limits the total possible inequality, thus confining the possible results of the inequality indexes.

With respect to method selection, we have used default parameters for I_{Atkinson} and I_{Kolm}, as discussed in Section 3.1. However, the choice of parameters influences the sensitivity of the two indices to transfers between the "rich" and the "poor", e.g., polluting the data by replacing a number of perfect IMs by the same number of imperfect IMs in a certain range. Therefore, a more extensive evaluation using different parameter values for I_{Atkinson} and I_{Kolm} is desirable.

Finally, we have considered data from a single version of Eclipse, and it is not clear whether the results can be generalized to different systems. Therefore, we need to consider a representative set of benchmarks, and a representative set of their versions.

8.4 Conclusions

One of the goals of software quality models is to warn software engineers and managers about potential problems with the software project. In this sense, the aggregation techniques for software metrics used in such quality models should highlight undesirable (bad) values in the aggregated result.

In this section we have empirically evaluated the satisfaction of this requirement of I_{Squale} and the inequality indices. We have performed experiments with data based on values of source lines of code measured per method in Eclipse 2.0, and we have studied the reaction of I_{Squale} and the inequality indices in the presence of an increasing large number of undesirable metrics values.

We have observed that I_{Squale} satisfies this requirement, and the choice of parameter λ enables the adjustment of the tolerance of I_{Squale} to such undesirable metrics values, with $\lambda = 30$ ensuring that the value of I_{Squale} drops as soon as very few undesirable values are present.

Regarding the suitability of the inequality indices for highlighting undesirable values in the aggregated result, we have observed that I_{Kolm} is the inequality index that best highlights bad results (i.e., low values), while I_{Theil} is the inequality index that least highlights bad results, even less than the arithmetic mean.

However, although I_{Kolm} seems the better choice if one is interested in highlighting undesirable values, it is also the inequality index least appropriate for this purpose whenever more than 30-40% undesirable values exist, since afterwards inequality as measured by I_{Kolm} starts to decrease. In contrast, the same problem does not become apparent for I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson} until the percentage of undesirable values exceeds 70%. Therefore, the latter inequality indices become more interesting than I_{Kolm} for the given purpose if the analyzed systems are assumed to have between 40-70% undesirable values.

All in all, inequality indices are generally inappropriate for highlighting undesirable results unless assumptions about the number of bad values can be made, since they do not discern between all values being equal but high, and all values being equal but low. Nonetheless, given also the close relation between I_{Squale} and I_{Kolm} discussed in Section 3.2, it might be interesting to study how the inequality indices can be better adapted to the needs of software quality assessments.

Concluding remarks

In the previous five chapters we have presented the results of a series of empirical studies of the main traditional, econometric, and threshold-based aggregation techniques for code metrics.

In order to assess the feasibility of performing large-scale studies of aggregation of code metrics, as well as to distil requirements for appropriate tooling to facilitate such studies, we first performed pilot studies. In these studies, discussed in Chapter 5, we aggregated SLOC values from class to package level on ArgoUML, Adempiere, and Mogwai Java Tools, and studied statistical correlation between the aggregated values and the *number of defects per package*, as well as statistical correlation between the various aggregation techniques themselves.

As a result, we observed that the choice of aggregation technique influences the correlation between SLOC values measured per class and aggregated to package level, and the number of defects per package. However, bugs are typically underreported in the issue tracking systems [9], suggesting that subsequent studies with the number of defects as validation metric are unfeasible, unless data quality can be ensured. Moreover, we observed high and statistically significant correlation between (i) I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover}, and I_{Atkinson}, and similarly between (ii) I_{Kolm}, *mean*, *standard deviation*, and *variance* on the one hand, and (iii) *skewness* and *kurtosis* on the other hand, suggesting that values aggregated using these techniques convey the same information.

However, the pilot studies are subject to a number of significant threats to validity, including the choice of metric, the representativeness of the systems considered, and the representativeness of their versions. Consequently, in order to alleviate these threats, we performed an extensive correlation study of the traditional and econometric aggregation techniques for a number of size (SLOC, LOC, NOS, NOSt), low-variance (DIT, NOC), and limited-range (PBS, PLwC) code metrics on the 106 systems comprising the Qualitas Corpus [110], and we presented the results thereof in Chapter 6.

In comparison to the results of the pilot studies, we now observed that (i) holds for all metrics and all versions of the systems analyzed, i.e., the aggregated values obtained using I_{Gini}, I_{Theil}, I_{MLD}, I_{Hoover} and I_{Atkinson} always convey the same information. In contrast, (ii) was only partly confirmed,

since the correlation between *mean* and I_{Kolm} was high and statistically significant only for the size metrics, while the correlation between I_{Kolm} and *standard deviation* or *variance* was lower and less statistically significant for all metrics. Finally, (iii) was also only partly confirmed, i.e., values aggregated using *skewness* and *kurtosis* convey the same information only for the size metrics.

In addition, regarding the influence of the specific version of a system on the correlation between the various techniques, we observed that correlation between I_{Theil} and I_{Kolm} increases as the system size increases for the size metrics, but has inconsistent behaviour for the other metrics, while correlation between *mean* and I_{Kolm} fluctuates without clear relation to system size.

Later, in Chapter 7, we changed the aggregation level from class–package to method–class in order to enable the comparison with the threshold-based aggregation techniques, and we presented the results of a correlation study of all three categories of aggregation techniques with focus on the threshold-based ones, using SLOC as metric. We observed that Squale and the discrete version of SIG show negative average correlation with other aggregation techniques, and positive average correlation with each other. In other words, there is redundancy in the information conveyed by values aggregated using Squale or discrete SIG, but both measure different characteristics than any of the other aggregation techniques. Moreover, we observed that changing the aggregation level with respect to the antecedent studies does not affect the correlation between the inequality indices, as measured in Chapter 6.

Finally, aggregation techniques which are part of software quality models (e.g., Squale) are usually designed to highlight undesirable metrics values in order to warn the software engineers in case of potential problems. In Chapter 8 we empirically compared the inequality indices to the aggregation technique in Squale in their ability to highlight undesirable values in the aggregated result. We observed that inequality indices are generally much less suited for highlighting undesirable values than Squale, unless assumptions about the number of bad values can be made, since they do not discern between all values being equal but high, and all values being equal but low.

Part III

Requirements

Chapter 9

Requirements for aggregation of software metrics

Based on insights derived from the theoretical (discussed in part I) and empirical (discussed in part II) analyses of traditional, econometric, and threshold-based aggregation techniques for code metrics, we propose requirements for future aggregation techniques for code metrics. These requirements will be categorized as "must", "should" and "could" to illustrate their varying importance (cf. [106]).

Note that we are still operating under the distinction between aggregation performed on values obtained by applying different metrics to the same software artifacts (e.g., the Maintainability Index), and aggregation performed on values obtained by applying the same metric to different software artifacts (e.g., average number of lines of code). Since throughout this thesis we focus on the latter, we also classify requirements as "must", "should" or "could" accordingly.

Must

- *Aggregation*: Must aggregate low level results of code metrics (from the level of individual software components like classes or methods) at a higher level (e.g., a subsystem or an entire project). All aggregation techniques considered in this thesis satisfy *aggregation*.
- *Symmetry*: The final result must not be dependent on the order of the elements being aggregated, i.e., it should be invariant to permutations of the individual values. All aggregation techniques considered in this thesis satisfy *symmetry*.

- *Domain compatibility*: The domain of the aggregation technique must be compatible with the range of the metrics being aggregated. For example, I_{Theil}, I_{MLD}, and I_{Atkinson} cannot be applied to negative values. Additionally, our experiments suggested that it is unfeasible to apply I_{Squale} to aggregation of large metrics values since the larger the value, the smaller the influence it has on the aggregated results (I_{Squale} is nominally applied to individual marks, i.e., normalized metrics values in the range $[0, 3]$).

- *Robustness*: Our experiments with existing aggregation techniques have shown that their behaviour may depend on the metric being aggregated, the system size, or the version of the system being analyzed (e.g., I_{Kolm} conveys the same information as the mean for size metrics). Therefore, we require that aggregation techniques be robust to different metrics, systems of different sizes, and different versions of the same system.

Should

- *Domain size*: The domain of the aggregation technique should be as big as possible to ensure compatibility with as many different metrics as possible. Note that the weaker *domain compatibility* requirement above does not necessarily guarantee compatibility with as many different metrics as possible if the range of the compatible metrics is small.

- *Highlight problems*: Should be more sensitive to problematic values in order to pinpoint them, and also to provide a stronger positive feedback when problems are corrected. For example, I_{Squale} satisfies this requirement, while the inequality indices only do so as long as assumptions about the number of undesirable values can be made (Chapter 8).

- *Decomposability*: An important question in interpreting the value aggregated on a system level pertains to the extent to which the result can be attributed to differences between system subcomponents: are the low level quality results concentrated only in few subcomponents or are they spread over the entire system? To answer this question it should be possible to express the aggregated result computed at a system level in terms of results computed for system subcomponents. Decomposability enables root-cause analyses (using I_{Theil}), as well as measuring to what extent the aggregated value at the system level can be explained by a specific partitioning of the system into subcomponents [29, 99] (e.g., with I_{Theil} or I_{Kolm}), as discussed in Section 3.2.

- *Reliable for skewed metrics*: As the distribution of many interesting code metrics is skewed [112], the interpretation of central tendency

measures (mean, median) becomes unreliable. The aggregation technique should be reliable under strongly-skewed distributions.

- *Correlation with mean*: Central tendency measures such as the mean become unreliable for strongly-skewed distributions commonly encountered with code metrics. Therefore, an aggregation technique which is *reliable for skewed metrics* should not statistically correlate with the mean.

- *Continuous aggregation range*: The aggregated result should be in a continuous range, to prevent thresholding (staircase) effects. In other words, the aggregation technique should reflect minor changes in individual values in order not to discouraging small, progressive improvements in quality.

Could

- *Normalized range*: The aggregated result could be normalized to a finite range, independent from the number of elements to be aggregated, in order to ease relative comparisons between different systems. For example, I_{Theil} ranges over $[0, \log n]$, where n is the number of elements to be aggregated, while I_{Squale} is normalized to the range $[0, 3]$.

- *Invariance with respect to addition*: If the aggregation technique is invariant with respect to addition (e.g., I_{Kolm}), then adding a constant to all individual values does not change the aggregated result. For example, the invariance with respect to addition allows to ignore, e.g., headers containing the licensing information and included in all source files.

- *Invariance with respect to multiplication*: If the aggregation technique is invariant with respect to multiplication (e.g., I_{Gini}), then multiplying all individual values by a constant does not change the aggregated result. For example, in case of aggregating source lines of code (SLOC) values measured per file with a multiplication-invariant aggregation technique, the results obtained are not affected if percentages of the total SLOC are considered rather than the SLOC values themselves.

  Note that the two invariance requirements are conflicting, thus cannot be satisfied simultaneously. Depending on the intended application, a choice must be made between one or the other.

- *Translatability*: means that adding a constant to all individual values increases the aggregated result by the same constant. For example, I_{Squale} satisfies *translatability*. Translatability might be interesting, e.g., for SLOC if the same header (containing licensing information) is added to all classes.

- *Transfers principle*: If the aggregation technique satisfies the transfers principle (e.g., I_{Kolm}, I_{Theil}), then a strictly positive transfer from a larger to a smaller individual value, without reversing their relative ordering and with leaving all other values unchanged, should result in a strictly positive reduction in the aggregated result. In contrast, one can talk about the "anti-transfers" principle (satisfied by I_{Squale}), meaning that the above-mentioned transfer should result in a strictly positive increase in the aggregated result.

  For example, if source lines of code values measured per method are considered undesirable when greater than 36, then a decrease in SLOC of 20 for one method with SLOC 60 at the cost of an equivalent increase in SLOC for another method with SLOC 20 would result in an increase of the aggregated result of an aggregation technique satisfying the "anti-transfers" principle (i.e., I_{Squale}).

- *Population principle*: If the aggregation technique satisfies the population principle, then the aggregated result is invariant with respect to replications of the population of individual values a finite number of times. I_{Squale}, traditional aggregation techniques such as the mean, and all inequality indices satisfy the population principle. It still has to be a subject of a further study whether the population principle is relevant for software metrics.

- *Composition*: The aggregation technique could be used on values obtained by applying different metrics to the same software artifacts. Note that with existing techniques, *composition* and *aggregation* of metrics are not necessarily mutually exclusive, and often the former encompasses the latter (e.g., the Maintainability Index). Therefore, composition could be performed either as a part of the same aggregation technique, or as a separate aggregation technique complementary to the first one.

- *Correlation with Squale or SIG*: Aggregation techniques that satisfy the *Highlight problems* requirements could statistically correlate with the Squale or SIG approaches, specifically designed to highlight potential problems in the aggregated result. For both Squale and SIG, our experiments have suggested that there is redundancy between them, but both measure different characteristics than any of the other aggregation techniques.

- *Correlation with I_{Theil}, I_{MLD}, or I_{Atkinson}*: These inequality indices satisfy most requirements identified in this section (e.g., *symmetry*, *decomposability*, *transfers principle*, *population principle*), therefore an aggregation technique satisfying these requirements could statistically correlate to I_{Theil}, I_{MLD}, or I_{Atkinson}.

Chapter 10

Conclusions

A popular approach to assessing software maintainability and predicting its evolution involves collecting and analyzing code metrics. As metrics are usually defined on a micro-level, and should provide insights in the evolution at the macro-level, the metrics values should be aggregated. Three main groups of aggregation techniques can be found in the literature on software metrics: traditional aggregation techniques such as the *mean*, *median*, or *sum*, and more recent econometric aggregation techniques, such as the *Gini*, *Theil*, *Kolm*, *Atkinson*, and *Hoover* inequality indices, and threshold-based aggregation techniques, such as the Squale and SIG approaches. However, a profound comparison of different aggregation techniques was, so far, missing.

In this thesis we focused on several main econometric and threshold-based aggregation techniques for code metrics with the purpose of deriving requirements for future aggregation techniques for software metrics. In this sense, we performed studies along two directions. In the first part we performed a theoretical analysis and studied properties of these techniques relevant to aggregation of code metrics. Our main contributions were proofs showing that root-cause analyses can be performed efficiently using I_{Theil}, and that the aggregation technique in Squale shares common properties with inequality indices, and has a formal relation to I_{Kolm}.

In the second part we presented the results of empirical studies of all three categories of aggregation techniques considered (i.e., traditional, econometric, and threshold-based) using different metrics and aggregation levels. First, we performed pilot studies, analyzing statistical correlation between the aggregation techniques and the number of defects, as well as between the aggregation techniques themselves. As a result, we concluded that subsequent studies with the number of defects as validation metric are infeasible unless data quality can be ensured. In addition, we proposed a methodology to perform empirical studies of aggregation techniques for software metrics, and appropriate tooling to support such studies.

Second, while concentrating on statistical correlation between the various aggregation techniques, we observed consistently high and statistically significant correlation between I_{Gini}, I_{Theil}, I_{MLD}, I_{Atkinson}, and I_{Hoover} for all metrics and all versions considered, i.e., aggregation values obtained using these techniques always convey the same information.

Third, we investigated the nature of the relation between various aggregation techniques. We noted that superlinear (e.g., between I_{Theil} and I_{Gini}), as well as chaotic (e.g., between I_{Theil} and I_{Kolm}) patterns can be observed in the scatter plots. This led to the observation that some indices may be more appropriate than others depending on which dimension of inequality one is interested in emphasizing, the choice of metric, or the intended application.

Fourth, we observed that changing the aggregation level to class level does not affect the correlation between various aggregation techniques as measured at package level. However, system size does influence the correlation between aggregation techniques, e.g., correlation between I_{Theil} and I_{Kolm}, as well as between I_{Atkinson} and I_{Kolm} increases as the system size increases, while correlation between *mean* and I_{Kolm} fluctuates without clear relation to system size.

Fifth, we observed redundancy in the information conveyed by values aggregated using Squale or discrete SIG, although both measure different characteristics than any of the other aggregation techniques considered.

Finally, we observed that inequality indices are generally inappropriate for highlighting undesirable values in the aggregated result unless assumptions about the number of bad values can be made, since they do not discern between all values being equal but high, and all values being equal but low.

As a consequence of both the theoretical and the empirical analyses performed, in the third part we concluded the thesis by proposing requirements for aggregation techniques for software metrics.

10.1 Future work

We consider a number of directions for *future work*, in addition to the ones already identified in each chapter.

Empirical studies

The empirical presented in this thesis can be furthermore extended by considering additional software metrics. Specifically, we intend to investigate the Chidamber and Kemerer suite [22] and the Lorenz and Kidd suite [65], as well as *metrics with negative values* such as MI [83].

Additionally, it is not clear whether the results obtained can be generalized to other software metrics, and to non-software domains. Therefore, we intend to replicate the studies to non-code metrics such as function points [2] and work effort metrics [119].

New aggregation techniques

In this thesis we have studied several main aggregation techniques for code metrics, i.e., the econometric inequality indices and the SIG and Squale threshold-based aggregation techniques. However, both categories of aggregation techniques lack desirable properties. For example, the inequality indices are not well suited for highlighting undesirable values in the aggregated results, while high ratings obtained with the threshold-based aggregation techniques are not necessarily an indication of good software engineering practices. The work presented in this thesis and the requirements for aggregation techniques for software metrics identified in Chapter 9 can serve as basis for designing new aggregation techniques for software metrics.

Aggregation vs. combination

In this thesis we focused on aggregation performed on values obtained by applying the same metric to different software artifacts (e.g., average number of lines of code) rather than aggregation performed on values obtained by applying different metrics to the same software artifacts (e.g., the Maintainability Index).

However, single metrics are rarely enough to characterize the maintainability of a software system. Therefore, metrics should often be combined. Metrics combination could be performed at different levels, e.g., at low level (for each component), or at a higher level (for the entire project), on already aggregated metrics. However, it is not clear which approach should be preferred, although in practice it is probably more meaningful to combine metrics at the lower level. For example, the *comment rate* quality evaluation would already be less meaningful at the level of a class than at the level of individual methods, since a class could have a very complex, poorly commented method and a very simple, over-documented one, resulting in an undesired, globally-good evaluation for the comment rate.

So far we have only applied inequality indices to aggregation of values obtained by applying the same metric to different software artifacts. Therefore, we intend to study the applicability of inequality indices to aggregation of combined metrics data.

Root-cause analyses

First, in Section 3.2 we have shown that root-cause analyses are possible using I_{Theil}, and can be performed efficiently. However, I_{MLD}, I_{Atkinson}, and I_{Kolm} are also decomposable, hence we intend to study their potential for root-cause analyses as well. Second, an extensive empirical validation of root-cause analyses using inequality indices is desirable. Therefore, we plan to extend the tooling presented in Appendix A in order to incorporate root-cause analyses, and perform empirical evaluation and validation studies.

Decomposability

Decomposability of I_{Theil}, I_{MLD}, I_{Atkinson}, and I_{Kolm} assumes the existence of a MECE partitioning of the system into subgroups, and allows computing the value of the inequality index at system level in terms of the inequality beween and within the subgroups.

Alternatively, one might be interested in determining a partitioning such that a certain property is satisfied. For example, when selecting a representative set of benchmarks for a certain software system, two conflicting needs should be satisfied, i.e., the collection should be restricted to be similar enough to the given system, while being large enough to be representative. We intend to apply inequality indices to study the influence of different characteristics of the systems on determining the set of representative benchmarks.

Social organization of software projects

In addition to empirical studies of software artifacts, a deeper understanding into open-source software evolution can be gained by also analyzing the social organization of software projects, i.e., the distribution of activity across the different members, which was showed to satisfy the Pareto principle [84]. When looking for evidence of the Pareto principle in the social organization of software projects or other areas of software engineering, inequality indices have been shown to provide useful insights [46]. However, a more extensive application of inequality indices to studying the interplay of software artifacts with the different project members that communicate and collaborate in order to construct and evolve the software is desirable.

List of Figures

List of Tables

Bibliography

[1] Takahiro Akita, Rizal Affandi Lukman, and Yukino Yamada. Inequality in the distribution of household expenditures in Indonesia: A Theil decomposition analysis. *Developing Economies*, XXXVII(2):197–221, June 1999.

[2] Allan J. Albrecht. Measuring Application Development Productivity. In I. B. M. Press, editor, *IBM Application Development Symp.*, pages 83–92, October 1979.

[3] Robert B. Allen and David Garlan. A formal approach to software architectures. In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing*, volume 1, pages 134–141, Amsterdam, The Netherlands, 1992. North-Holland Publishing Co.

[4] Paul D. Allison. Measures of inequality. *American Sociological Review*, 43(6):865–880, 1978.

[5] Tiago L. Alves, José P. Correia, and Joost Visser. Benchmark-based aggregation of metrics to ratings. Available online at `http://wiki.di.uminho.pt/twiki/pub/Personal/Tiago/Publications/alves2011-draft.pdf` (accessed June 2011), 2011.

[6] Tiago L. Alves, Christiaan Ypma, and Joost Visser. Deriving metric thresholds from benchmark data. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[7] Francis John Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, 1973.

[8] Anthony Barnes Atkinson. On the measurement of inequality. *Journal of Economic Theory*, 2(3):244–263, 1970.

[9] Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 97–106. ACM, 2010.

[10] Françoise Balmas, Fabrice Bellingard, Simon Denier, Stéphane Ducasse, Bertrand Franchet, Jannik Laval, Karine Mordal-Manet, and Philippe Vaillergues. *The Squale Quality Model. Modèle enrichi d'agrégation des pratiques pour Java et C++*. INRIA, 2010.

[11] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, pages 4–17, 2002.

[12] Henrike Barkmann, Rüdiiger Lincke, and Welf Löwe. Quantitative evaluation of software quality metrics in open-source projects. In *Advanced Information Networking and Applications (WAINA'09). International Conference on*, pages 1067–1072. IEEE, 2009.

[13] Victor R. Basili. Software modeling and measurement: the Goal/Question/Metric paradigm. Technical report, College Park, MD, USA, 1992.

[14] Gareth Baxter, Marcus Frean, James Noble, Mark Rickerby, Hayden Smith, Matt Visser, Hayden Melton, and Ewan Tempero. Understanding the shape of Java software. *ACM SIGPLAN Notices*, 41(10):397–412, 2006.

[15] Ahmed Belderrar, Segla Kpodjedo, Yann-Gaël Guéhéneuc, Giulio Antoniol, and Philippe Galinier. Sub-graph mining: Identifying micro-architectures in evolving object oriented software. In *CSMR 2011: 15th European Conference on Software Maintenance and Reengineering*, pages 171–180. IEEE, 2011.

[16] Charles Blackorby, David Donaldson, and Maria Auersperg. A new procedure for the measurement of inequality within and among population subgroups. *The Canadian Journal of Economics/Revue canadienne d'Economique*, 14(4):665–685, 1981.

[17] Francois Bourguignon. Decomposable income inequality measures. *Econometrica*, 47(4):901–20, July 1979.

[18] Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51(3):245–273, 2000.

[19] Frederick P. Brooks Jr. *The mythical man-month: essays on software engineering*, volume 7. Addison-Wesley, 1995.

[20] Giacomo Bucci, Fabrizio Fioravanti, Paolo Nesi, and Sandro Perlini. Metrics and tool for system assessment. In *iceccs*, page 0036. Published by the IEEE Computer Society, 1998.

[21] David Gawen Champernowne. A comparison of measures of inequality of income distribution. *The Economic Journal*, 84(336):787–816, 1974.

[22] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20:476–493, June 1994.

[23] Giulio Concas, Michele Marchesi, Sandro Pinna, and Nicola Serra. Power-laws in a large object-oriented software system. *IEEE Trans. Software Eng.*, 33(10):687–708, 2007.

[24] Pedro N. Conceicao and Pedro M. Ferreira. The Young Person's Guide to the Theil Index: Suggesting Intuitive Interpretations and Exploring Analytical Applications. *SSRN eLibrary*, 2000.

[25] John D. Cooper and Matthew J. Fisher. *Software quality management.* Petrocelli Books, 1979.

[26] Anna Corazza, Sergio Di Martino, Valerio Maggio, and Giuseppe Scanniello. Investigating the use of lexical information for software system clustering. In *CSMR 2011: 15th European Conference on Software Maintenance and Reengineering*, pages 35–44. IEEE, 2011.

[27] Jose Pedro Correia and Joost Visser. Certification of technical quality of software products. In *Proc. of the Int'l Workshop on Foundations and Techniques for Open Source Software Certification*, pages 35–51, 2008.

[28] Frank A. Cowell. Inequality decomposition: three bad measures. *Bulletin of Economic Research*, 40(4):309–312, 1988.

[29] Frank A. Cowell. Measurement of inequality. volume 1 of *Handbook of Income Distribution*, pages 87 – 166. Elsevier, 2000.

[30] Frank A. Cowell and Stephen P. Jenkins. How much inequality can we explain? A methodology and an application to the United States. *Economic Journal*, 105(429):421–30, 1995.

[31] Frank A. Cowell and K. Kuga. Inequality measurement: An axiomatic approach. *Eur. Econ. Review*, 15(3):287–305, 1981.

[32] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[33] Christian Damgaard and Jacob Weiner. Describing inequality in plant size or fecundity. *Ecology*, 81(4):1139–1142, 2000.

[34] James Davies and Michael Hoy. Making inequality comparisons when lorenz curves intersect. *The American Economic Review*, 85(4):980–986, 1995.

[35] Jean-Yves Duclos. Social evaluation functions, economic isolation and the suits index of progressivity. *Journal of Public Economics*, 69(1):103–121, 1998.

[36] Jean-Yves Duclos and Abdelkrim Araar. An Atkinson-Gini family of social evaluation functions. *Economics Bulletin*, 3(19):1–16, 2003.

[37] Marc Eaddy, Thomas Zimmermann, Kaitlin D. Sherwood, Vibhav Garg, Gail C. Murphy, Nachiappan Nagappan, and Alfred V. Aho. Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, 34:497–515, July 2008.

[38] Kalhed El Emam, Saïda Benlarbi, Nishith Goel, and Shesh N. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Trans. Softw. Eng.*, 27:630–650, 2001.

[39] Norman E. Fenton and Shari Lawrence Pfleeger. *Software metrics: a rigorous and practical approach*. PWS Publishing Co., 1998.

[40] James E. Foster. An axiomatic characterization of the Theil measure of income inequality. *Journal of Economic Theory*, 31(1):105–121, 1983.

[41] Mark Gabel and Zhendong Su. A study of the uniqueness of source code. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 147–156. ACM, 2010.

[42] Tom Gilb. *Software metrics*. Winthrop computer systems series. Winthrop Publishers, 1977.

[43] Corrado Gini. Variabilitè e mutabilitè. *Studi Econornico-Giuridici della R. Universita de Cagliari*, 1912.

[44] Nils Göde and Jan Harder. Clone stability. In *CSMR 2011: 15th European Conference on Software Maintenance and Reengineering*, pages 65–74. IEEE, 2011.

[45] Bindu Goel and Yogesh Singh. Empirical investigation of metrics for fault prediction on object-oriented software. *Computer and Information Science*, pages 255–265, 2008.

[46] Mathieu Goeminne and Tom Mens. Evidence for the Pareto principle in Open Source Software Activity. In *Proc. Int'l Workshop SQM 2011*. CEUR-WS, 2011.

[47] Gregory A. Hall and John C. Munson. Software evolution: code delta and code churn. *Journal of Systems and Software*, 54(2):111–118, 2000.

[48] Klaus Marius Hansen, Kristján Jónasson, and Helmut Neukirchen. An empirical study of open source software architectures' effect on product quality. Technical Report VHI-01-2009, Engineering Research Institute, University of Iceland, July 2009. `http://www.hi.is/~kmh/doc/vhi-01-2009.pdf`.

[49] Petra Heck, Martijn Klabbers, and Marco van Eekelen. A software product certification model. *Software Quality Journal*, 18(1):37–55, 2010.

[50] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, pages 30 –39, 2007.

[51] Brian Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, 1996.

[52] Israel Herraiz. A statistical examination of the evolution and properties of libre software. In *ICSM*, pages 439–442. IEEE, 2009.

[53] Israel Herraiz, Jesus M. Gonzalez-Barahona, and Gregorio Robles. Towards a theoretical model for software growth. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 21. IEEE, 2007.

[54] Edgar Malone Hoover Jr. The measurement of industrial localization. *The Review of Economic Statistics*, 18(4):162–171, 1936.

[55] Darrell Huff. *How to lie with statistics*. WW Norton, New York, 1954.

[56] ISO/IEC. Iso/iec 9126-3 software engineering -product quality- part 3: Internal metrics, 2003.

[57] Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[58] Barbara A. Kitchenham, Lesley M. Pickard, Stephen G. MacDonell, and Martin J. Shepperd. What accuracy statistics really measure [software estimation]. In *Software, IEE Proceedings-*, volume 148, pages 81–85. IET, 2001.

[59] Serge-Christophe Kolm. The Optimal Production of Social Justice. In *Public Economics*, pages 145–200. MacMillan, London, 1969.

[60] Serge-Christophe Kolm. Unequal inequalities I. *Journal of Economic Theory*, 12(3):416–442, 1976.

[61] Robert G. Lanergan and Charles A. Grasso. Software engineering with reusable designs and code. *IEEE Trans. Softw. Eng.*, (5):498–501, 1984.

[62] Eric Langford, Neil Schwertman, and Margaret Owens. Is the property of being positively correlated transitive? *The American Statistician*, 55(4):322–325, 2001.

[63] Michele Lanza and Radu Marinescu. *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, 2006.

[64] Skylar Lei and Michael R. Smith. Evaluation of several nonparametric bootstrap methods to estimate confidence intervals for software metrics. *IEEE Trans. Softw. Eng.*, 29:996–1004, 2003.

[65] Mark Lorenz and Jeff Kidd. *Object-oriented software metrics: a practical guide*. Prentice-Hall, 1994.

[66] Max O. Lorenz. Methods of measuring the concentration of wealth. *Publications of the American Statistical Association*, 9(70):209–219, 1905.

[67] Mark Lutz and David Ascher. *Learning Python*. O'Reilly Media, 2004.

[68] John H. Maindonald and John Braun. *Data analysis and graphics using R: an example-based approach*, volume 10 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 2007.

[69] Spiros Mancoridis, Brian S. Mitchell, Yih-Farn Chen, and Emden R. Gansner. Bunch: a clustering tool for the recovery and maintenance of software system structures. In *Int. Conf. on Softw. Maintenance*, pages 50–59, 1999.

[70] Radu Marinescu. *Measurement and Quality in Object-Oriented Design*. PhD thesis, "Politehnica" University of Timisoara, Romania, 2002.

[71] Albert W. Marshall, Ingram Olkin, and Barry C. Arnold. *Inequalities: theory of majorization and its applications*. Springer, 2005.

[72] Robert Martin. OO design quality metrics: An analysis of dependencies, 1994. Available at `http://condor.depaul.edu/~dmumaugh/OOT/`

`Design-Principles/oodmetrc.pdf` Consulted on January 11, 2009.

[73] Thomas J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, pages 308–320, 1976.

[74] Thomas J. McCabe and Charles W. Butler. Design complexity measurement and testing. *Communications of the ACM*, 32(12):1415–1425, 1989.

[75] Jim A. McCall, Paul K. Richards, and Gene F. Walters. *Factors in Software Quality*. NTIS Springfield, 1976.

[76] Karine Mordal-Manet, Jannik Laval, Stéphen Ducasse, Nicolas Anquetil, Françoise Balmas, Fabrice Bellingard, Laurent Bouhier, Philippe Vaillergues, and Thomas J. McCabe. An empirical model for continuous and weighted metric aggregation. In *15th Eur. Conf. Soft. Maintenance and Reeng.*, pages 141–150. IEEE, 2011.

[77] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*, pages 181–190. ACM, 2008.

[78] Paul Murrell. *R graphics*. Computer Science and Data Analysis Series. CRC Press, 2006.

[79] Nachiappan Nagappan, Thomas Ball, and Andres Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[80] Gottfried Emanuel Noether. Why Kendall tau? *Teaching Statistics*, 3(2):41–43, 1981.

[81] Hector M. Olague, Letha H. Etzkorn, Sampson Gholston, and Stephen Quattlebaum. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on software Engineering*, pages 402–419, 2007.

[82] Paul Oman and Jack Hagemeister. Metrics for assessing a software system's maintainability. In *Software Maintenance, 1992. Proceerdings., Conference on*, pages 337–344. IEEE, 1992.

[83] Paul Oman and Jack Hagemeister. Construction and testing of polynomials predicting software maintainability. *Journal of Systems and Software*, 24(3):251–266, 1994.

[84] Vilfredo Pareto. *Manuale di economia politica*. Societa Editrice, 1906.

[85] Simon C. Parker. The inequality of employment and self-employment incomes: a decomposition analysis for the U.K. *Review of Income and Wealth*, 45(2):263–274, 1999.

[86] Karl Pearson. Note on Regression and Inheritance in the Case of Two Parents. *Royal Society Proceedings*, 58:240–242, 1895.

[87] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, and Zahir Tari. Coupling metrics for predicting maintainability in service-oriented designs. In *18th Australian Softw. Eng. Conf.*, pages 329–340, April 2007.

[88] Shari Lawrence Pfleeger. Software metrics: Progress after 25 years? *Software, IEEE*, 25(6):32–34, 2008.

[89] Shari Lawrence Pfleeger, Ross Jeffery, Bill Curtis, and Barbara A. Kitchenham. Status report on software measurement. *Software, IEEE*, 14(2):33 –43, 1997.

[90] Boris A. Portnov and Daniel Felsenstein. Measures of regional inequality for small countries. In Boris A. Portnov and Daniel Felsenstein, editors, *Regional Disparities in Small Countries*, chapter 4, pages 47–62. Springer Verlag, College Station, Texas, 2005.

[91] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C/C++: The Art of Scientific Computing Code*. Cambridge University Press, 2002.

[92] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010.

[93] Paula Rooney. Microsoft's ceo: 80-20 rule applies to bugs, not just features. `http://www.crn.com/news/security/18821726/microsofts-ceo-80-20-rule-applies-to-bugs-not-just-features.htm`, retrieved June 2011, 2002.

[94] Xavier Sala-i-Martin. The world distribution of income: Falling poverty and convergence, period. *The Quarterly Journal of Economics*, 121(2):351–397, 2006.

[95] Ioannis Samoladas, Georgios Gousios, Diomidis Spinellis, and Ioannis Stamelos. The sqo-oss quality model: Measurement based open source software evaluation. *Open Source Development, Communities and Quality*, pages 237–248, 2008.

[96] Katherine V. Schinasi. Defense acquisitions: Stronger management practices are needed to improve dod's software-intensive weapon acquisitions. Technical Report GAO-04-393, Government Accountability Office Washington DC, 2004.

[97] Amartya K. Sen and James E. Foster. *On economic inequality*. Oxford University Press, USA, 1973.

[98] Alexander Serebrenik, Serguei Roubtsov, and Mark G. J. van den Brand. D_n-based architecture assessment of Java open source software systems. In *ICPC '09: Proc. 17th Int. Conf. on Program Comprehension, 2009*, pages 198–207. IEEE, 2009.

[99] Alexander Serebrenik and Mark G. J. van den Brand. Theil index for aggregation of software metrics values. In *Int. Conf. on Software Maintenance*, pages 1–9. IEEE, 2010.

[100] Martin Shepperd and Gada Kadoda. Using simulation to evaluate prediction techniques [for software]. In *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*, pages 349–359. IEEE, 2001.

[101] Anthony F. Shorrocks. The class of additively decomposable inequality measures. *Econometrica*, 48(3):613–625, 1980.

[102] Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In *29th Euromicro Conference*, pages 336–342, September 2003.

[103] Luciana Silva, Klérisson Paix ao, Sandra de Amo, and Marcelo Maia. On the use of execution trace alignment for driving perfective changes. In *CSMR 2011: 15th European Conference on Software Maintenance and Reengineering*, pages 221–230. IEEE, 2011.

[104] Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, pages 441–471, 1987.

[105] Murray R. Spiegel and Larry J. Stephens. *Schaum's outline of theory and problems of statistics*. Teach Yourself, 2008.

[106] Jennifer Stapleton. DSDM: Dynamic Systems Development Method. In *Proceedings of the Technology of Object-Oriented Languages and Systems*, page 406. IEEE Computer Society, 1999.

[107] Perdita Stevens. A landscape of bidirectional model transformations. In Ralf Lämmel, Joost Visser, and João Saraiva, editors, *GTTSE*, volume 5235 of *Lecture Notes in Computer Science*, pages 408–424. Springer, 2007.

[108] Giancarlo Succi, Witold Pedrycz, Snezana Djokic, Paolo Zuliani, and Barbara Russo. An empirical exploration of the distributions of the Chidamber and Kemerer object-oriented metrics suite. *Empirical Softw. Eng.*, 10:81–104, January 2005.

[109] Ewan Tempero. Qualitas Corpus 20101126 release notes. `http://qualitascorpus.com/docs/history/20101126.html`, 2010.

[110] Ewan Tempero, Craig Anslow, Jens Dietrich, Ted Han, Jing Li, Markus Lumpe, Hayden Melton, and James Noble. Qualitas corpus: A curated collection of java code for empirical studies. In *Asia Pacific Software Engineering Conference*, 2010.

[111] Henri Theil. *Economics and Information Theory*. North-Holland, 1967.

[112] Ivana Turnu, Giulio Concas, Michele Marchesi, Sandro Pinna, and Roberto Tonelli. A modified Yule process to model the evolution of some object-oriented system properties. *Inf. Sci.*, 181:883–902, February 2011.

[113] Rajesh Vasa, Markus Lumpe, Philip Branch, and Oscar M. Nierstrasz. Comparative analysis of evolving software systems using the Gini coefficient. In *Int. Conf. on Software Maintenance*, pages 179–188. IEEE, 2009.

[114] Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. Comparative study of software metrics' aggregation techniques. In *9th Belgian-Netherlands Softw. Evolution Seminar*, pages 80–84, Lille, 2010.

[115] Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. By no means: A study on aggregating software metrics. In *2nd International Workshop on Emerging Trends in Software Metrics*, Honolulu, Hawaii, USA, 2011.

[116] Bogdan Vasilescu, Alexander Serebrenik, and Mark G. J. van den Brand. You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics. In *Int. Conf. on Software Maintenance*. IEEE, 2011. accepted.

[117] Dieter Welzel and Hans-Ludwig Hausen. Practical concurrent software evaluation for certification. *Journal of Systems and Software*, 38(1):71–83, 1997.

[118] Richard Wheeldon and Steve Counsell. Power law distributions in class relationships. In *Source Code Analysis and Manipulation, 2003.*

Proceedings. Third IEEE International Workshop on, pages 45–54, September 2003.

[119] Karl E. Wiegers. Lessons from software work effort metrics. *Software Development*, page 46, 1994.

[120] Achim Zeileis. *ineq: Measuring Inequality, Concentration, and Poverty*. R Foundation for Statistical Computing, 2009.

[121] Achim Zeileis. Package 'ineq' for R. Technical report, CRAN, 2009.

Appendix A

Proposed tooling

In order to meet the requirements defined in Section 5.5 and perform the empirical evaluation of different aggregation techniques, we propose tooling adhering to the *pipes-and-filters* architectural style [3], i.e., a sequence of processing steps, each performing a specific function.

The motivation for the pipes-and-filters architectural style is twofold. First, a pipes-and-filters architecture facilitates performing many transformations and being flexible in using them, without sacrificing robustness. In this sense, filters (e.g., extracting new metrics or aggregating metrics using new aggregation techniques) can be easily added, omitted, or rearranged into a new sequence without having to change the filters themselves.

Second, by dividing the processing into simpler specialized transformations, the complexity of individual filters is lowered, making them easier to implement and to test, and improving their reusability. Even though the communication between processing steps increases, resulting in increased latency and overhead, the benefits of reusing individual transformations outweighs the drawbacks.

Since the tooling is meant as a research prototype, thus only as a means towards an end, we chose Python as the implementation language, due to the speed of development in it [67]. The complete tooling totals approximately 3000 SLOC, distributed across 12 files. The development effort is estimated to around 25 man-days.

The tooling consists of a number of Python modules, implementing specialized transformations (e.g., package name extractor, parser for Understand metrics, boxplot generator), and a number of Python scripts, each corresponding to one of the different experiments described in Chapters 5–8, which assemble the different previously-developed modules into a series of processing steps that together represent a certain experiment.

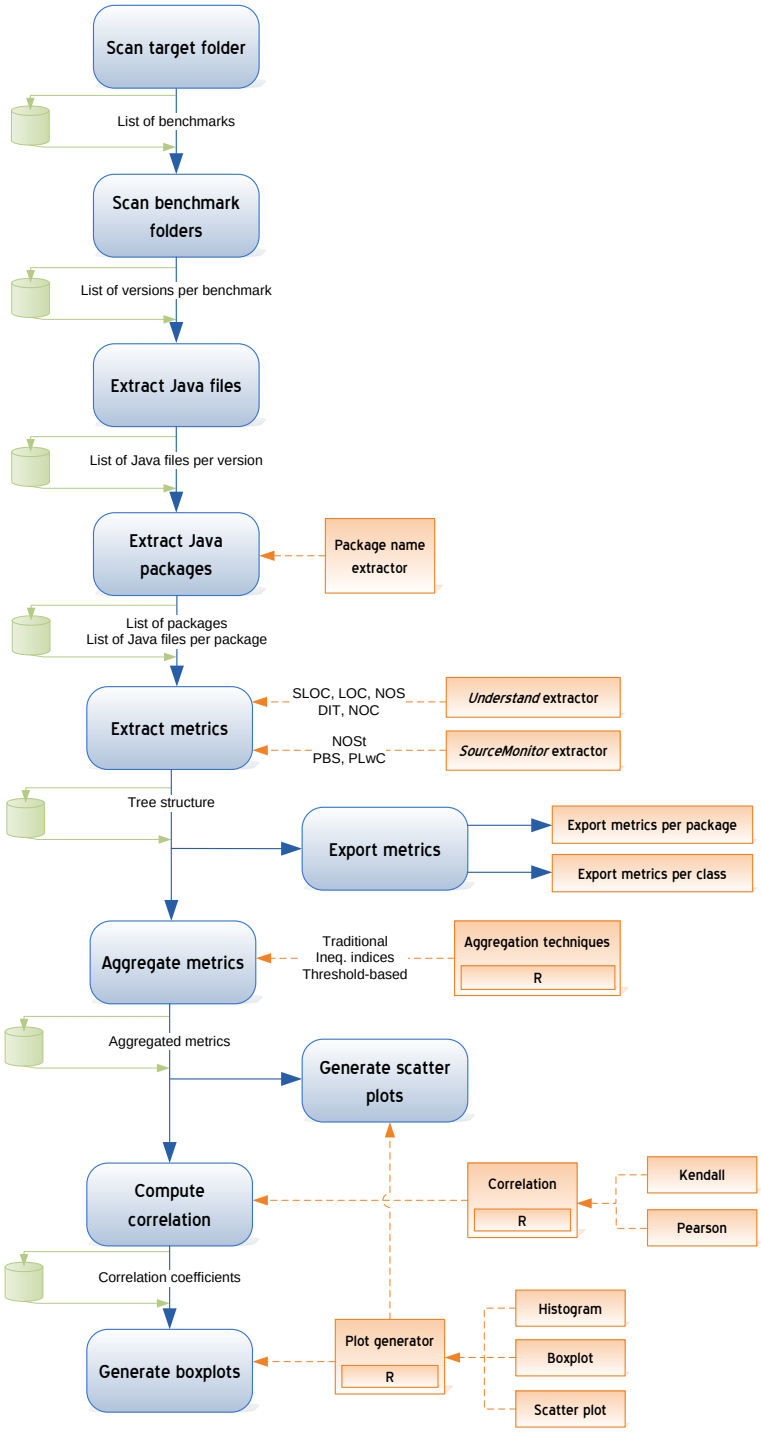

Figure A.1: Example workflow for aggregation of software metrics that allows extensions to the pilot studies in Chapter 5, as identified in Section 5.5.

A.1 Example workflow

We illustrate the proposed tooling with the example workflow in Figure A.1, designed to easily accommodate extensions to the pilot studies in Chapter 5 along the directions identified in Section 5.5, i.e., metrics, representative sets of benchmarks and their versions, and threshold-based aggregation techniques.

The tooling is designed around the Qualitas Corpus [110], a curated collection of open-source Java software systems, intended to be used for empirical studies of code artifacts. The contents of the Corpus is organized into a hierarchical structure, in which *Systems* is the top-level directory containing a subdirectory *sysname* for every system in the corpus. Then, each such directory *sysname* contains only subdirectories *sysname-version_id* corresponding to each version of the system in the Corpus. For each version of each system in the corpus, the source code is distributed in the *src* subdirectory of *sysname-version_id*.

The example workflow in Figure A.1 starts by scanning the target (installation) folder of the Corpus, determining which systems are present, and extracting the list of Java files in each version. This list is then fed to *Package name extractor* which generates a list of Java packages per version of a system, and a list of the Java classes *directly* contained in those packages. Recall from Section 5.2.1 that we say that a class C is directly contained in a package P if there exists no subpackage P' of P different from P such that C is contained in P'. Between two consecutive processing steps, the output of the first step is exported to disk such that intermediate results could be reused for different purposes if necessary, and steps could be skipped if subsequent experiments would not affect their output.

Computing the actual metrics data is performed using third-party tools. In the current version, our tooling implements two metrics data extractors from the output files of Understand[1] and SourceMonitor[2], but additional third-party tools can be easily supported by implementing the necessary interfaces. Regardless of the exact extractor, the metrics data is stored in an internal tree structure organized corresponding to the package structure, which enables experimentation with different aggregation levels depending on how the tree is traversed. For example, in the study presented in Chapter 6 we aggregate metrics data from class to package level, while in the study described in Chapter 7 we aggregate metrics data from method to class level. To enable external processing, the tooling also implements export modules for the internal metrics data to CSV format, at different aggregation levels (e.g., class- or package-level). If one is interested in aggregation levels

[1] `http://www.scitools.com`

[2] `http://www.campwoodsw.com/sourcemonitor.html`

that are orthogonal to the package structure, then she can use, e.g., model transformations [32, 107] to convert the tree to the desired form.

Aggregation of the metrics data is performed in a separate transformation, which uses wrappers for R [92] around each aggregation technique. Integration with R is desirable since R provides good library support for statistical computing (e.g., all traditional and econometric aggregation techniques are readily available), and good visualizations [68, 78] Extending the tooling to include additional aggregation techniques is thus easily enabled by providing a wrapper for an R implementation if available, or the actual implementation otherwise. Several transformations can be performed on the aggregated data. For example, statistical correlations can be computed between pairs of metrics data values aggregated using different aggregation techniques. Statistical correlation coefficients (e.g., Pearson, Kendall) are computed using wrappers for R implementations, which can be easily extended. Alternatively, our tooling provides wrappers for various R plots (e.g., histograms, boxplots, or scatter plots), which can be used during any transformation step. The example workflow in Figure A.1 illustrates two such uses of plot generation, i.e., scatter plots to study the nature of the relation between various aggregation techniques (cf. Figure 6.11), and boxplots to study how consistent correlation coefficients are across the entire Corpus (cf. Figure 6.1).

A.2 Conclusions

In this chapter we have presented Python tooling conforming to the pipes-and-filters architectural style, designed as a research prototype for the empirical evaluation of different aggregation techniques. By adhering to the pipes-and-filters architectural style, the proposed tooling facilitates separation of concerns, division of labor, specialization, and reuse. The proposed tooling satisfies the requirements identified in Section 5.5, as follows:

- Flexibility: by decoupling extraction of metrics data, aggregation, and analysis, and by using a tree structure for the representation of the metrics data, the proposed tooling is easily extendible to new metrics and aggregation techniques, enabling aggregation of metrics values at different levels.

- Integration with metrics-extraction tools: the proposed tooling enables the further processing of metrics-data previously extracted by third-party tools. Currently two data extractors are implemented, for Understand and SourceMonitor metrics. An effort of around 400 SLOC is expected when implementing an extractor for a different tool.

- Reuse of existing components: the proposed tooling uses R for computing the aggregated values, as well as for generating different plots

with the results, e.g., histograms, boxplots, or scatter plots. Additionally, the proposed tooling uses Understand and SourceMonitor for computing metrics.

- Scalability: the tooling was designed around the Qualitas Corpus, a curated collection of open-source Java software systems, intended to be used for empirical studies of code artifacts. The Corpus comes in two main distributions, one containing the most recent versions available at the time of release from 106 systems, the other containing all versions from 13 systems (out of the 106 systems) with 10 or more versions available, totaling 414 versions. The two distributions of the Corpus total 434,771 files and 57,047,227 SLOC. The analysis time for the workflow described in Figure A.1 is approximately 4 hours, excluding the processing time of Understand and SourceMonitor, but including the processing time of R.

- Export: the proposed tooling exports data in CSV format.

- Research prototype: the proposed tooling is a research prototype implemented in Python, easily extendible due to the speed of development in Python [67], and the choice of architectural style [3].