International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 3, March 2015, PP 16-20
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

Aggregation Techniques on Software Metrics: A Study

S.V. Achuta Rao[1], R. Kiran Kumar[2]

[1]Research Scholar, [2]Computer Science Department
Krishna University, Machilipatnam, India
[1]sarachyuth@gmail.com, [2]kirankkreddir@gmail.com

Abstract: *Metrics are usually defined on a micro level like methods, classes and packages. These are failed to provide an adequate picture of the entire system effectively. By combine different metrics with varying output values and ranges to get insight in the evolution of the macro level system. We listed various metrics like Product, Project & Process and also various aggregation techniques in Traditional methods such as mean, median, sum, and cardinality; in Distribution fittings such as Log-Normal, Exponential, Negative binomial; and in Inequality Indices such as Theil, Gini, Kolm and Atkinson. The theoretical criteria in Domain, Range, and Invariance & Decomposability of various metrics are discussed. The Aggregation Techniques from simple mathematical operations to more complex operations to get Macro level system would be helpful as the outliers get pulled into the larger amounts of data. The Developers or Managers have an understanding of the parts of system are still needed to make sure that the metrics not misused or misunderstood. Metris are powerful tools that need to be used with care. We wish to understand, the aggregation techniques influence the strength of the relations among metrics to asses software quality.*

Keywords: *Software Metrics, Aggregation Techniques, Software Quality, Micro-Level & Macro-Level*

1. INTRODUCTION

A Quote on Measurement by Lord William Kelvin (1824 – 1907) like this "When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science." A software metric is a quantitative measure of a degree to which a software system process or posesses some property [7]. Since quantitative measurements are essential in all sciences, there is a continuous effort by computer science practitioners and theoreticians to bring similar approaches to software development. Metrics are defined earlier in Micro level include: Bugs per line of code, Comment Density, Code Coverage, Cohesion, Coupling, Complexity(McCabe's Complexity), DSQI (Design Structure Quality Index), Function Point Analysis, Halstead Complexity, Instruction Path Length, Maintainability index, Number of Classes and interfaces, Number of lines of code, Number of lines of customer requirements, Program execution time, Program Load time, Program size(binary), Program Execution time, Program Load time, Software Package metrics, Weighted Micro Function points, Function points and Automated Functional points, Object Management Group Standards, CISQ automated quality characteristics measures[3]. The metrics are not adequately characterized all the attributes of process, project and product requirements. Hence practitioners and theoreticians are started combined and aggregated metrics used on simple mathematical assessments.

The goal is obtaining objectives of macro level system which may have numerous valuable applications in schedule and budget planning, cost estimate, quality assurance testing, software debugging, software performance optimization influence the strength of the relations among metrics[2,3]. The aggregation of different metrics to obtain a single value helps in the global evaluation of a task, project, etc. This need also arises when different metrics are used. The more traditional aggregation techniques are additive or similar, namely mean, median, or sum. Sometimes these techniques are too crude to be entirely useful. We think that the aggregation methodology should be clear to the analyst, but at the same time sophisticated enough to represent the different aspects of the underlying metrics used and flexible enough so the model can be easily adapted to new quality assurance requirements[4]. Let us note that there are two sides to software metrics aggregation. One is when applying different metrics – or the same metric– at different levels of granularity of a given piece of software. The other is when applying different metrics – or the same metric – to different software artifacts intended for comparison purposes. In general all aggregation techniques apply to both. Apart from

the simplest strategies of metric aggregation, there are also a number of other methods for aggregation using different techniques, such as those using indexes or coefficients employed in other areas such as econometrics, e.g. Gini [1], Theil [2], coefficients or even the Paretto principle [1,2]. B. Vasilescu [3] analyzes several aggregation methods for the aggregation of software metrics to measure software quality. This is done from two points of view –first a theoretical analysis is done and then an empirical one is carried out. In [4] Mordal-Manet et al present not only the problem that metrics alone are not enough to characterize software quality but also an empirical model –the Squale model [4] – for metric aggregation. This model has four levels adding practices as an intermediate level between criteria and metrics that are the levels suggested in ISO 9126. For assessment purposes it uses an evaluation scale that falls in the interval [0:3], it uses a weighted average, and the function uses a constant to define hard, medium, or soft weighting. L. Etaati et al in [5] employ a Fuzzy Group Analytical Network Process method to integrate metrics to evaluate e-learning systems. This is a similar method to our proposal however it does not use Continuous Logic functions; moreover the network is not as easy to comprehend as the models obtained from the application of the LSP method. Bearing the above in mind we have as a main goal to aggregate the data obtained from different quality evaluation metrics in coherent groupings so as to get new singular values that can in turn be aggregated again. The aggregation ends getting a single global indicator for the software object under evaluation, being this objects a software unit or an entire software project.

To achieve this process we use operators from a Continuous Logic, specifically the Logic employed by the LSP method that proposes the aggregation of preferences by using a group of logic functions called Generalized Conjunction Disjunction (GCD) operators. So we show here a model –based on the ISO/IEC 9126 international standard [7] – to aggregate software quality metrics employing a Continuous Logic[6]. This standard establishes a number of requirements to evaluate software quality, however there is no prescription for the aggregation of the different measurements proposed. Therefore, there exists the need to propose an aggregation model to obtain a single value out the evaluation with different metrics.

2. TRADITIONAL AGGREGATION TECHNIQUES

The traditional aggregation techniques are explained in two stages. First, the earlier combined quality metrics and their elements are explained. Second, classical aggregationl techniques such as mean, median, sum, and cardinality; Distribution fittings and in Inequality Indices such as Theil, Gini, Kolm and Atkinson and theoretical criteria in Domain, Range, Invariance & Decomposability of various metrics are discussed[3,4,5].

2.1 Combining Different Metrics

Aggregation of software metrics can be understood in two ways. First, there is a need to combine di_erent metrics as recommended by quality-model design methods such as Factor-Criteria-Metric (FCM) [5], or Goal-Question-Metric (GQM) [6], 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 [4].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. Example 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. aggregate each metric from method / class level to the system level, and then combine these system-level results into a unified assessment. Here 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 [7].

2.2 Classical Aggregation Techniques On Metrics

In The Theoretical comparison, we study a number of mathematical properties of the aggregation techniques relevant for their application to software metrics.

Domain. Domain of the aggregation technique determines applicability of this technique to classes of software metrics. Econometric indices are usually applied to income or welfare distributions, i.e., to sets of positive values. Some software metrics, however, may have negative values, e.g., the maintainability index [7]. Since $\log z$ and $\sqrt{z}$ are undefined for $z < 0$, ITheil and IAtkinson are undefined as well. Unlike these indices, the mean, IGini and IKolm can be used to aggregate negative values. Moreover, as log 0 is undefined direct application of the Theil index

formula from is not possible. However, *I*Theil(*x*1, . . . , *xn*−1, *xn*) can be defined for *xn* = 0 depending on whether zero denotes emptiness (e.g., SLOC, number of classes in a package) or not. All other aggregation techniques considered in this paper can be applied to zero values. Finally, formulas for the Gini index, the Theil index and the Atkinson index involve division by $\bar{x}$. Hence, these indices are undefined if $\bar{x} = 0$. The mean and the Kolm index do not have additional cases when their values are undefined.

Range: Interpretation of the aggregated value depends on the range of the aggregation technique: e.g., 0.99 indicates a very high degree of inequality if *I*Gini is considered, while in case of *I*Theil and *I*Atkinson the interpretation would depend on the number of values being aggregated. The values obtained by applying the mean can range from −∞ to +∞. The Gini index is often claimed to range over [0, 1] [2]: this is, however, the case only if all the values being aggregated are positive.In general, this is not necessarily the case: *I*Gini(1, −1.5) = −2.5. Range of *I*Theil and *I*Atkinson depends on the number of values being aggregated: one can show that 0 ≤ *I*Theil(*x*1, . . . , *xn*) ≤ log *n* and 0 ≤ *I*Atkinson(*x*1, . . . , *xn*) ≤ 1 − 1*n* . The Kolm index ranges over non-negative reals.

Invariance: We say that the aggregation technique is invariant with respect to addition if *I*(*x*1, . . . , *xn*) = *I*(*x*1 + *c*, . . . , *xn* + *c*) for any *x*1, . . . , *xn* and *c*, provided *I*(*x*1 + *c*, . . . , *xn* + *c*) exists. Similarly, we say that the aggregation technique is invariant with respect to multiplication if *I*(*x*1, . . . , *xn*) = *I*(*x*1*c*, . . . , *xn c*) for any *x*1, . . . , *xn* and *c*, provided *I*(*x*1*c*, . . . , *xn c*) exists. Aggregating lines of code measured per file, aggregation-technique-invariant with respect to addition allows to ignore, e.g., headers containing the licensing information and included in all source files. Results obtained by applying an aggregation technique that is invariant with respect to multiplication are not affected if percentages of the total number of lines of code are considered rather than the number of lines of code themselves.The mean is neither invariant with respect to addition nor to multiplication. It can be shown that *I*Gini , *I*Theil and *I*Atkinson are invariant with respect to multiplication. Unlike these indices, *I*Kolm is invariant with respect to addition. *Decomposability.* Decomposability 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[5]. Formally, *I* is decomposable if for any given partition {*x*1,1, . . . , *x*1,*n*1, . . . , *xJ*,1, . . . , *xJ*,*nJ* } of {*x*1, . . . , *xn*} it holds that *I*(*x*1, . . . , *xn*) = *I*($\bar{x}$1, . . . , $\bar{x}$*J*) + _*J j*=1 (*wj* ∗ *I*(*xj*,1, . . . , *xj*,*nj*)) for some coefficients *w*1, . . . ,*wJ* satisfying _*Jj* =1 *wj* = 1, where $\bar{x}j$ is the mean of *xj*,1, . . . , *xj*,*nj* . Then the ratio of the inequality between the groups and the total amount of inequality can be seen as the percentage of inequality that can be explained by partitioning the population into groups. Both *I*Theil [1] and *I*Kolm [2] are decomposable, while *I*Gini and *I*Atkinson are not [3]. It should be noted that while some authors propose means of decomposing *I*Gini or *I*Atkinson, they use a slightly different notion of decomposability [2 ,3].

The *Gini index*, the *Theil index*, the *Kolm index* and the *Atkinson index* have already been applied to software metrics in [2, 3], respectively.

IGini(X) = 12n2_xPni=1Pnj=1 jxi □ xj j

*I*Gini(*x*1, . . . , *xn*) = 12n^{-}_*x*_*ni*=1_*n j*=1|*xi* − *xj* [1]

ITheil(X) = 1nPni=1□ xi_x log xi_x_

*I*Theil(*x*1, . . . , *xn*) = 1*n*_*n i*=1_*xi*$^{-}$*x* log *xi*$^{-}$*x*_ [2]

IAtkinson(X) = 1 □ 1_x□ 1nPni=1pxi_2

*I*Atkinson(*x*1, . . , *xn*) = 1 − 1^{-}*x*_1*n*_*ni*=1√*xi*_2 [3]

IKolm(X) = log_ 1nPni=1 e_x□xi

*I*Kolm(*x*1, . . . , *xn*) = log_1*n*_*ni*=1 *e*$^{-}$*x*−*xi*_[4]

3. REQUIREMENTS IN COMPOSITION / AGGREGATION ON METRICS

Composition: Metrics used to assess a practice can be composed, e.g., by: – Simple or weighted averaging of the different values of the metrics. This is only possible when the different metrics have similar range and semantic; – Thresholding on one metric such as cyclomatic complexity to consider or not the other metrics, for example, when cyclomatic complexity is more than 50, one could decide to divide the number of lines of comment by some value to highlight the fact that overly complex methods need to be overly commented; – Interpolating, given examples components by the developers and their perceived evaluation of quality (e.g., one method with 50 LOC would be perceived of quality 2.5 —on an interval of [0, 3]— and another example with 100 LOC would be perceived of quality 1.5), one can interpolate a function to convert other values; The result of the composition of metrics values for a practice is called Individual Mark (IM). Individual marks for a practice are computed from raw metrics with multiple ranges, and constitute single marks in the range [0, 3]. The raw metrics composed may have multiple ranges.

- Aggregation: Aggregation of IMs for a practice requires several steps (illustrated with an example in Figure 1; the dark dots on the x-axis are the IMs to be aggregated–0.5, 1.5, and 3): 1. A weighting function is applied to each IM: g(IM) =

λ −IM where IM is the individual mark and λ the constant defining the "hard", "medium", or "soft" weighting. Hard weighting gives more weight to bad results than soft weighting. λ is greater for a hard weighting and smaller for a soft one .

Theorem 1: Let x1, . . . , xn be real numbers and let x^{-} = 1 n Pn i=1 xi .

Then for λ > 1 min(x1, . . . , xn) ≤ I λ Squale(x1, . . . , xn) ≤ x.

Proof: Since min(x1, . . . , xn) ≤ xi for all 1 ≤ i ≤ n, then it also holds that −xi ≤ − min(x1, . . . , xn).

Since λ > 1 it holds that λ −xi ≤ λ − min(x1,...,xn) for all i. Therefore, Pn i=1 λ −xi ≤ nλ− min(x1,...,xn) ≡ 1 n Pn i=1 λ −xi ≤ λ − min(x1,...,xn) ≡ logλ 1 n Pn i=1 λ −xi ≤ − min(x1, . . . , xn) ≡ min(x1, . . . , xn) ≤ I λ Squale(x1, . . . , xn) Now, the geometric mean never exceeds the arithmetic mean, i.e., pn Qn i=1 λ−xi ≤ 1 n Pn i=1 λ −xi . However, pn Qn i=1 λ−xi = λ − 1 n Pn i=1 xi = λ −x^{-} . Hence, λ −x^{-} ≤ 1 n Pn i=1 λ −xi Since λ > 1, −x^{-} ≤ logλ 1 n Pn i=1 λ −xi ≡ − logλ 1 n Pn i=1 λ −xi ≤ x^{-} ≡ I λ Squale(x1, . . . , xn) ≤ x "Must" requirements are imposed by our perception of low-level metric values' combination as a sequence of two steps, composition and aggregation; "should" and "could" requirements were based on properties of aggregation techniques found in the literature .

Must:

- *Aggregation:* Must aggregate low level quality results (from the level of individual software components like classes or methods) at a higher level (e.g., a subsystem or an entire project) to evaluate the quality of an entire project, as discussed
- *Composition*: Must compose different metric values with different ranges to a single quality interval, as explained in
- *Composition/Aggregation Range and Domain:* Whether composition occurs before aggregation or the opposite, the range (output) of the first must be compatible with the domain (input) of the second. For example, if the aggregation formula contains a logarithm, the composition method must have strictly positive range;

Should:

- 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, as discussed in §2; • Do not hide progress: Improvement in quality should never result in a worsening of the evaluation [1,2]. As a counter example, it is known that econometric inequality indices will worsen when going from an "all equally-bad" situation to a situation where all are equally bad except one;
- Decomposability: Should be decomposable in order to measure to what extent the aggregated value at the system level can be explained by a specific partitioning of the system into subsystems [4,5,6];
- Composition before Aggregation

Composition before Aggregation: Composition should be performed at the level of individual components to retain the intended semantic of the composition;

- *Aggregation range:* Should be in a continuous scale, preferably bounded (i.e., left and right-bounded) ;
- *Symmetry:* The final result should not be dependent on the order of the elements being aggregated. This requirement is typically not applicable for composition, since, for example, one can hardly expect a composition function f defined on size s and cyclomatic complexity v to satisfy f(s, v) = f(v, s);

Could:

- *Evaluation normalization:* Could normalize all results (metrics, combination, aggregation) to allow unified interpretation at all levels
- *Invariance and translatability:* Both invariance and translatability are interesting, e.g., for SLOC, if the same header (containing licensing information) is added to all classes (invariance with respect to addition and translatability), or if percentages of the total SLOC are considered rather than the number itself (invariance with respect to multiplication).

4. CONCLUSION

There are numerous software quality metrics available to measure the varying aspect of the quality of software, these metrics are defined at a low level of individual components: functions, methods, classes, whereas developers need a global view at the level of an entire system. But this should not be an issue in practice because it is unlikely to occur. Because there is an important literature on econometric indexes, it might be interesting to continue studying them and see how they can be adapted to the needs of quality assessment. We suggest one area of research, noticing that the experiments that the distribution of quality results for individual components is limited to two small intervals whereas in real life they could be much more spread out.

The aggregation of software quality metrics study on both traditional and econometric aggregation techniques, applied techniques c should be considered. Furthermore, we find a need to investigate the nature of the relation between various

aggregation techniques like linear (e.g., between ITheil and IAtkinson), superlinear (e.g., between ITheil and IGini), as well as chaotic (e.g., between

ITheil and IKolm) 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.

Classical aggregation techniques have problems when distributions are skewed. Inequality indices look more promising.

REFERENCES

[1] *S. M. Metev Karine Mordal, Nicolas Anquetil, Jannik Laval, Alexander Serebrenik, Bogdan , Vasilescu and Stéphane Ducasse,Technische Universiteit , The Netherlands, Software quality metrics aggregation in industry , J. Softw. Evol. and Proc.(2012)*

[2] *B. Vasilescu, "Analysis of Advanced Aggregation Techniques for Software Metrics". Master thesis,Eindhoven University of Technology, Department of Mathematics and Computer Science. Eindhoven, Netherlands, July 2011.*

[3] *A. Serebrenik, M. van den Brand, "Theil index for aggregation of software metrics values" ICSM, pages 1–9, IEEE Computer Society, 2010.*

[4] *Bogdan Vasilescu, Alexander Serebrenik, Mark van den Brand Technische Universiteit Eindhoven, The Netherlands, You Can't Control the Unfamiliar: A Study on the Relations Between Aggregation Techniques for Software Metrics Software quality metrics aggregation in industry, J. Softw. Evol. and Proc.(2012)*

[5] *Eric Bouwers, Software Improvement Group, The Netherlands, Joost Vissser, Software Improvement Group, Towards a catalog format for software metrics, Software Engineering Group Technical Report: WETSom 2014..*

[6] *Aristides Dasso, Ana Funes ,Software Engineering Group, Universidad Nacional de San Luis, Ejército de los Andes 950,San Luis, Argentina; Software Quality Metrics Aggregation, 13th Argentine Symposium on Software Engineering, ASSE 2012*

[7] *K. Mordal-Manet et al., "An empirical model for continuous and weighted metric aggregation". 2011 15th European Conference on Software Maintenance and Reengineering. March 1–4, 2011, Oldenburg, Germany.*

[8] *www.en.wikipedia.org/wiki/Software_Metricsl*

[9] *www.en.wikipedia.org/wiki/Software_quality*