RTTOOL: A Tool for Extracting Relative Thresholds for Source Code Metrics

Paloma Oliveira, Fernando P. Lima
Department of Computing
IFMG-Formiga, Brazil
{paloma.oliveira,fernando.lima}@ifmg.edu.br

Marco Tulio Valente
Department of Computer Science
UFMG, Brazil
mtov@dcc.ufmg.br

Alexander Serebrenik
Eindhoven University of Technology
The Netherlands
a.serebrenik@tue.nl

***Abstract*—Meaningful thresholds are essential for promoting source code metrics as an effective instrument to control the internal quality of software systems. Despite the increasing number of source code measurement tools, no publicly available tools support extraction of metric thresholds. Moreover, earlier studies suggest that in larger systems significant number of classes exceed recommended metric thresholds. Therefore, in our previous study we have introduced the notion of a relative threshold, *i.e.*, a pair including an upper limit and a percentage of classes whose metric values should not exceed this limit.**

In this paper we propose RTTOOL, an open source tool for extracting relative thresholds from the measurement data of a benchmark of software systems. RTTOOL is publicly available at http://aserg.labsoft.dcc.ufmg.br/rttool.

***Index Terms*—Source code metrics; Relative thresholds; Software quality; Software measurement.**

I. INTRODUCTION

Software metrics have been around since the dawn of software engineering. Well-known source code metrics include, e.g., cyclomatic complexity, number of attributes (NOA), number of methods (NOM), response for a class (RFC), number of other classes referenced by a class (FAN-OUT) and weighted method count (WMC) [1, 2, 3]. To promote the use of metrics as an effective measurement instrument to decision making, it is essential to establish meaningful thresholds [4, 5, 6]. In this way, for example, software quality managers can rely on such thresholds to assess the maintainability of software systems.

Most metric thresholds proposed in literature are based on experience or vision of what constitutes desirable software properties. For instance, industrial code standards for Java recommend that classes should have no more than 20 methods and that methods should have no more than 75 lines of code [7]. However, many source code metrics are known to follow heavy-tailed distributions [6, 8]. Therefore, it is not surprising that many studies report extremely high metric values violating any preconceived or suggested thresholds [9, 10].

We argue that metric thresholds should accommodate those outlier values. Specifically, in our previous work we advocated *relative thresholds*, stating that at least $p\%$ of the system entities should not exceed a threshold k [4]. Moreover, we proposed a corpus-based extraction algorithm for automatic derivation of relative thresholds. In the current paper, we present RTTOOL, a tool supporting the proposed algorithm. We illustrate the usage of RTTOOL by deriving relative thresholds for four metrics based on 106 open-source Java systems from the Qualitas Corpus [11]. To assess the performance of RTTOOL we also derived the thresholds for 20 metrics based on systems from the Qualitas Corpus.

RTTOOL is applicable to any software metric measured at the level of a class as long as low(er) metric values are considered to be more desirable than the high(er) ones, and the metrics distribution is heavy-tailed. Numerous metrics including NOA, NOM, FAN-OUT, RFC and WMC satisfy these conditions [4, 12, 13]. Examples of a metric that does not follow the traditional heavy-tailed distribution and therefore should not be subject to RTTOOL are DIT (Depth of Inheritance) [6] and D_n [14].

RTTOOL is independent of the exact way the metric values are calculated. Indeed, importance of differences between tools calculating "the same metrics" has been observed in the past [15]: RTTOOL does not take a stance in this debate. Moreover, RTTOOL can be configured for different contexts, *e.g.*, system size or application domain, since context is known to be crucial when deriving metrics thresholds [16].

RTTOOL also indicates the systems with outlier behavior, *i.e.*, systems that do not adhere the relative thresholds. It also generates several partial results, *e.g.*, plot of the cumulative density function, to examine the distribution of the values of a metric and identify relative thresholds for a given projects' collection. Those thresholds can be used to benchmark new projects or to monitor evolution of the existing ones.

The remainder of this paper is structured as follows. Section II summarizes our previous work on deriving relative thresholds [4]. Section III presents the design and implementation of the RTTOOL, Section IV illustrates the use of the RTTOOL by means of a case study, Section V discusses related work and Section VI presents final remarks.

II. RELATIVE THRESHOLDS

This section presents our method to extract relative source code metric thresholds [4]. We focus on software metrics that follow heavy-tailed distributions, when measured at the level

$$
\begin{aligned}
\mathit{ComplianceRate}[p,k] &= \frac{|\ \{\ S \in \mathit{Corpus} \mid p\%\ \text{of the classes in } S \text{ have } M \le k\}\ |}{|\ \mathit{Corpus}\ |} \\
\mathit{penalty}_1[p,k] &= \begin{cases} \dfrac{90 - \mathit{ComplianceRate}[p,k]}{90} & \text{if } \mathit{ComplianceRate}[p,k] < 90 \\ 0 & \text{otherwise} \end{cases} \\
\mathit{penalty}_2[k] &= \begin{cases} \dfrac{k - \mathit{Median90}}{\mathit{Median90}} & \text{if } k > \mathit{Median90} \\ 0 & \text{otherwise} \end{cases} \\
\mathit{ComplianceRatePenalty}[p,k] &= \mathit{penalty}_1[p,k] + \mathit{penalty}_2[k]
\end{aligned}
$$

Fig. 1. *ComplianceRate* and *ComplianceRatePenalty* functions [4]

of classes as long as low(er) metric values are considered to be more desirable than the high(er) ones. Our goal is to derive *relative thresholds*, *i.e.*, pairs $[p, k]$ such that $p\%$ *of the classes should have* $M \le k$, where M is a given source code metric. This relative threshold tolerates, therefore, $(100 - p)\%$ of classes with $M > k$.

Figure 1 presents the functions used to calculate the parameters p and k that define the relative threshold for a given metric M. Function $\mathit{ComplianceRate}[p, k]$ returns the percentage of systems in the *Corpus* that follows the relative threshold defined by the pair $[p, k]$. To determine the best p and k our approach tries to maximize *ComplianceRate* with a minimal *ComplianceRatePenalty*, where *ComplianceRatePenalty* is the sum of penalties introduced as follows:

- A $\mathit{ComplianceRate}[p, k]$ less than 90% receives a penalty proportional to its distance to 90%, as defined by function $\mathit{penalty}_1[p, k]$. This penalty fosters the selection of thresholds followed by at least 90% of the systems in the *Corpus*.
- A $\mathit{ComplianceRate}[p, k]$ receives the second penalty proportional to the distance between k and the median of the 90-th percentiles of the values of M in each system in the *Corpus*, denoted *Median90*, as defined by function $\mathit{penalty}_2[k]$.

III. RTTool: An Overview

The execution of the RTTool is divided into three stages: configuration, processing, and presentation (Figure 2).

In the **configuration** stage, the user selects the dataset, with the metric values collected for a given corpus. Since metrics calculation is separate from relative thresholds derivation, RTTool does not depend on the way the metrics have been calculated, and can be used in conjunction with any metrics calculation tool. The current version of RTTool accepts CSV or XML files with metrics values as input.

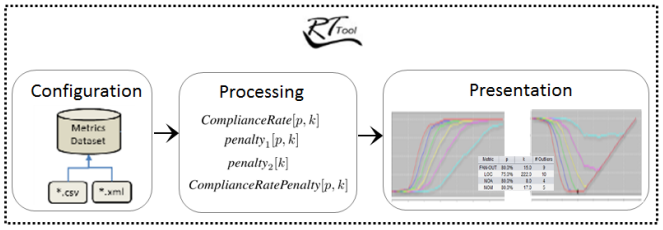

Fig. 2. RTTool stages

The **processing** stage is responsible for deriving the p and k values of the relative threshold. In this stage, RTTool also identifies the systems considered as outliers, i.e., systems that do not conform to the relative thresholds. More specifically, this stage calculates functions showed in Figure 1.

Finally, in the **presentation** stage, the results are shown as spreadsheets and graphs. The spreadsheets summarize the relative thresholds derived and the outlier systems identified (Figure 4). The presentation stage also plots a number of graphs including $\mathit{ComplianceRate}[p, k]$, and $\mathit{ComplianceRatePenalty}[p, k]$ (Figures 5 and 6, respectively).

IV. Evaluation

A. Example of usage

In order to illustrate the usage of our tool, we derive relative thresholds for a number of metrics collected for the 106 systems in Qualitas Corpus (version 20101126r) [11]. We used the Moose platform [17] and VerveineJ[1] to compute the values of the metrics for each class of each system and store them as CSV files.

First, to use the RTTool, the user must select the metrics to extract relative thresholds. After uploading the CSV files generated by Moose, 20 metrics become available for analysis (Figure 3) and the user selects four of them: FAN-OUT, NOA, LOC, and NOM.

Then, RTTool calculates the p and k values, that characterize relative thresholds for each metrics and shows the number and the names of the outlier systems (Figure 4). We can observe that the p values derived for different metrics are close suggesting that the k thresholds derived hold for 75%-80% of the systems. Moreover, we see that Weka, HSQLDB, and JTOpen appear as outliers for at least three metrics.

Finally, by inspecting Figures 5 and 6 we can see how RTTool has selected the relative thresholds. Indeed, by inspecting Figure 6 we observe that 80% is the highest value of p such that there exists k satisfying $\mathit{ComplianceRatePenalty}[p, k] = 0$.

[1] http://www.moosetechnology.org/tools/verveinej

This k equals 15 and is denoted with a small black circle on Figure 6. By consulting Figure 5 we observe that 90% of the *Corpus* systems follow the relative threshold $[80\%, 15]$ derived.

Using the slide bar on the right the user can select the p values she would like to inspect. The value on the slide bar indicates the lowest value to be visualized together with the curves obtained for p with increments of 5%. As expected, relaxing the relative threshold p value *e.g.*, to 70% results in a lower k equals 10 and in a comparable *ComplianceRate* of 82% (Figure 5).

Fig. 3. Configuration window

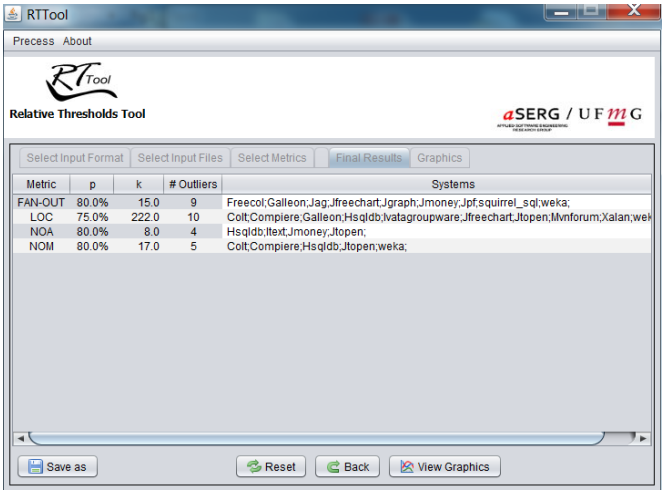

Fig. 4. Final results — with thresholds and outliers systems for each metric

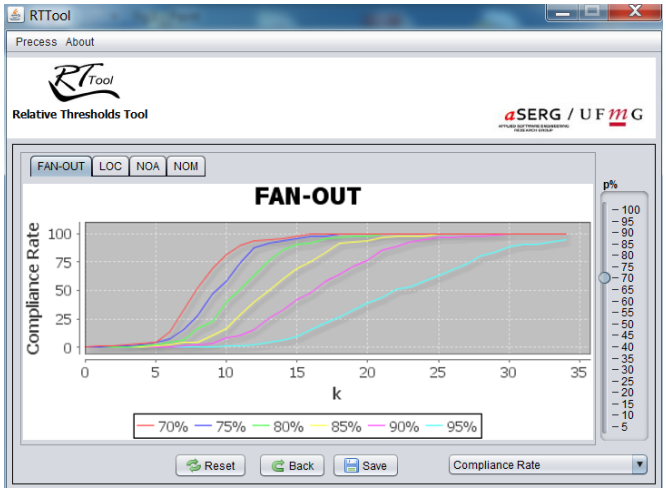

Fig. 5. ComplianceRate function (FAN-OUT metric)

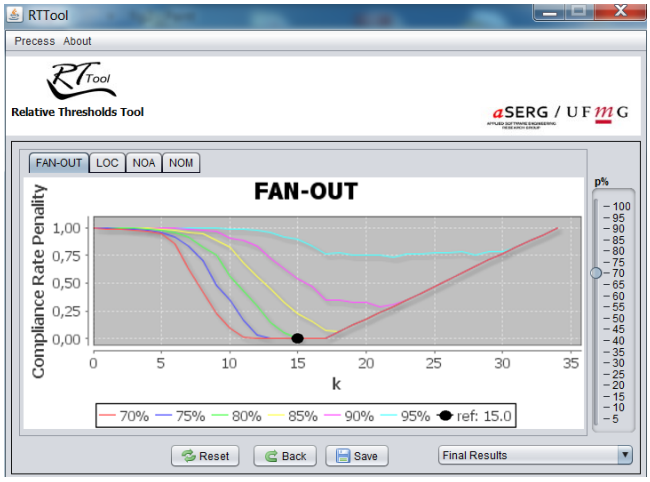

Fig. 6. ComplianceRatePenalty function (FAN-OUT metric)

B. Performance

To evaluate the performance of RTTOOL we have measured the runtime in four experiments by varying the size of the corpus, *i.e.*, the entire Qualitas Corpus (106 systems) vs. Qualitas Corpus systems classified as Tools by the corpus curators (27 systems), and the number of metrics (four metrics selected as in the example above vs. all twenty metrics available in the dataset). Table I summarizes the runtime measurements as reported by RTTOOL itself. The experiments have been run on a device with core i5 processor and 4GB DDR3 memory.

As expected, increasing the number of metrics or the size of the corpus results in higher execution times. However, even for the largest corpus and the maximal number of metrics the calculation time remains acceptable, slightly exceeding one minute (75949 milliseconds).

C. Availability

RTTOOL is an open-source project, distributed under the MIT license. We have opted for the MIT license since it permits

TABLE I
RUNTIME OF RTTOOL

Corpus	# systems	# metrics	time (ms)
Qualitas Corpus—Tools	27	4	13753
Qualitas Corpus—Tools	27	20	35056
Qualitas Corpus	106	4	15867
Qualitas Corpus	106	20	75949

reuse of the source code in the proprietary software: in this way we hope that the relative threshold calculation implemented in RTTOOL can find a way both to mainstream metrics calculation tools [18] and to research prototypes focusing on software analytics [19]. The proposed tool is available at http://aserg.labsoft.dcc.ufmg.br/rttool.

V. RELATED WORK

Extraction of thresholds based on system corpus has been studied in the literature [6, 16, 20, 21]. Unfortunately, most of these approaches are not supported by tools, the work of Alves, Ypma and Visser [16] being the only notable exception. The tool proposed in this work focuses on extracting absolute thresholds, weights the metrics based on LOC of the corresponding entities and constructs four quality profiles corresponding to low risk (0 to 70th percentiles), moderate risk (70th to 80th percentiles), high risk (80th to 90th percentiles), and very-high risk (90th percentile). As opposed to this line of work, RTTOOL derives relative thresholds, does not perform weighting and considers only two "quality profiles" (adhering to the relative thresholds or not). Finally, the tool of Alves, Ypma and Visser is proprietary, while RTTOOL is open source.

Rather then deriving thresholds, absolute or relative, distinguishing between "good" and "bad" metrics values, a number of papers suggest to evaluate distribution of metrics values by means of more advanced aggregation techniques [13, 22, 23, 24].

VI. CONCLUSIONS

In this paper we describe RTTOOL, an open-source tool capable of extracting relative thresholds for software metrics based on benchmark collections. Uniqueness of the tool is due to the choice for *relative thresholds*, *i.e.*, thresholds that should be followed by "most" of the system classes rather than all of them. The tool has been successfully applied to Qualitas Corpus, a well-known curated collection of open source Java systems, and its run-time performance has been evaluated in a series of experiments.

ACKNOWLEDGMENTS

Our research is supported by CAPES (Process Number: BEX 14468/13-1), FAPEMIG, and CNPq.

REFERENCES

[1] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[2] F. B. Abreu and R. Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in *4th International Conference of Software Quality*, 1994, pp. 3–5.

[3] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006.

[4] P. Oliveira, M. T. Valente, and F. Lima, "Extracting relative thresholds for source code metrics," in *CSMR-WCRE*, 2014, pp. 254–263.

[5] M. Foucault, M. Palyart, J.-R. Falleri, and X. Blanc, "Computing contextual metric thresholds," in *SAC*, 2014, pp. 1–10.

[6] K. Ferreira, M. Bigonha, R. Bigonha, L. Mendes, and H. Almeida, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, 2011.

[7] California Institute of Technology, "JPL institutional coding standard for the Java programming language," Tech. Rep., 2010.

[8] G. Baxter, M. Frean, J. Noble, M. Rickerby, H. Smith, M. Visser, H. Melton, and E. Tempero, "Understanding the shape of Java software," in *OOPSLA*, 2006, pp. 397–412.

[9] I. Herraiz, D. Rodriguez, and R. Harrison, "On the statistical distribution of object-oriented system properties," in *WETSOM*, 2012, pp. 56–62.

[10] J. Kaczmarek and M. Kucharski, "Size and effort estimation for applications written in Java," *Information and Software Technology*, vol. 46, no. 9, pp. 589–601, 2004.

[11] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, "The Qualitas corpus: A curated collection of Java code for empirical studies," in *APSEC*, 2010, pp. 336–345.

[12] R. Wheeldon and S. Counsell, "Power law distributions in class relationships," in *SCAM*, 2003, pp. 45–54.

[13] A. Serebrenik and M. G. J. van den Brand, "Theil index for aggregation of software metrics values," in *ICSM*, 2010, pp. 1–9.

[14] A. Serebrenik, S. A. Roubtsov, and M. G. J. van den Brand, "D_n-based architecture assessment of Java open source software systems," in *ICPC*, 2009, pp. 198–207.

[15] R. Lincke, J. Lundberg, and W. Löwe, "Comparing software metrics tools," in *ISSTA*, 2008, pp. 131–142.

[16] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *ICSM*, 2010, pp. 1–10.

[17] O. Nierstrasz, S. Ducasse, and T. Gîrba, "The story of Moose: an agile reengineering environment," *Software Engineering Notes*, vol. 30, no. 5, pp. 1–10, 2005.

[18] G. A. Campbell, P. P. Papapetrou, and O. Gaudin, *SonarQube in Action*. Manning Publications Company, 2013.

[19] M. G. J. van den Brand, S. A. Roubtsov, and A. Serebrenik, "SQuAVisiT: A flexible tool for visual software analytics," in *CSMR*, 2009, pp. 331–332.

[20] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics," *Journal of Empirical Software Engineering*, vol. 16, no. 6, pp. 812–841, 2011.

[21] R. Shatnawi, W. Li, J. Swain, and T. Newman, "Finding software metrics threshold values using ROC curves," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 1, pp. 1–16, 2010.

[22] B. Vasilescu, A. Serebrenik, and M. G. J. van den Brand, "You can't control the unfamiliar: A study on the relations between aggregation techniques for software metrics," in *ICSM*, 2011, pp. 313–322.

[23] ——, "By no means: a study on aggregating software metrics," in *WETSoM*, 2011, pp. 23–26.

[24] K. Mordal, N. Anquetil, J. Laval, A. Serebrenik, B. Vasilescu, and S. Ducasse, "Software quality metrics aggregation in industry," *Journal of Software: Evolution and Process*, vol. 25, no. 10, pp. 1117–1135, 2013.