Learning to Aggregate: An Automated Aggregation Method for Software Quality Model

Meng Yan[1], Xiaohong Zhang[1], Chao Liu[1], Jie Zou[1], Ling Xu[1], Xin Xia[2]
[1]School of Software Engineering, Chongqing University, Chongqing, China
[2]Department of Computer Science, University of British Columbia, Canada
Email: {meng.yan, xhongz, liu.chao, zoujie, xuling}@cqu.edu.cn, xxia02@cs.ubc.ca

***Abstract*—Quality models are regarded as a well-accepted approach for assessing high-level abstract quality characteristics (e.g., maintainability) by aggregation from low-level metrics. However, most of the existing quality models adopt the weighted linear aggregation method which suffers from a lack of consensus in how to decide the correct weights. To address this issue, we present an automated aggregation method which adopts a kind of probabilistic weight instead of the subjective weight in previous aggregation methods. In particular, we utilize a topic modeling technique to estimate the probabilistic weight by learning from a software benchmark. In this manner, our approach can enable automated quality assessment by using the learned knowledge without manual effort. In addition, we conduct an application on the maintainability assessment of the systems in our benchmark. The result shows that our approach can reveal the maintainability well through a correlation analysis with the changed lines of code.**

I. Introduction

Software quality model aims to provide the specific information about different high-level abstract quality characteristics (e.g., functionality, reliability, usability, efficiency, maintainability and portability defined in ISO/IEC 9126) by aggregation from the low-level concrete metrics of software products (e.g., number of source lines of code) [1]. However, decisions of the aggregation method for software quality model are rarely justified in this line of research. The most common used aggregation approach is weighted linear equations (WLE). Although this approach is simple to calculate and easy to interpret by practitioners, there is an issue which remains in the WLE method, i.e., how to decide the correct weights. A weight represents the relative importance contributed to the associated element in relation to its brother nodes. A usual way is to adopt empirical values or expert opinions by using Analytic Hierarchy Process. Unfortunately, software quality is a multifaceted and vague concept which has different meanings for different people [2]. Introducing the expert opinions, which depends on their experience, knowledge or intuition, may make the aggregation subjective [3] and prevent the model from being applied automatically.

To address this issue, we present an automated aggregation method which adopts a probabilistic weight learning from a benchmark instead of the subjective weight in previous aggregation methods. In particular, our approach is inspired by the success of transferring the generative topic model (e.g., Probabilistic Latent Semantic Analysis [4]) into different fields, including computer vision and software text mining [5], [6], [7], [8], [9]. The transferring power of the topic model derives from its fundamental purpose, namely, finding the probabilistic correlation between the hidden layer and the observed layers. For instance, the fundamental purpose of the topic model in text mining is to find the probabilistic correlation between the hidden topic and the observed words and documents. Similarly, we treat each code file as a document, each metric as a word and the quality of the characteristic as the "topic" hidden in the code file. Under this manner, we propose an automated aggregation method based on Probabilistic Latent Semantic Analysis (PLSA) which captures the hidden probabilistic correlation between quality characteristics, metrics, and code files by modeling from a benchmark. Subsequently, we construct "badness" function by adopting the probabilistic correlation to perform the aggregation step. As a result, we can assess the quality characteristics automatically and it overcomes the ambiguity and subjective interpretations from previous methods.

II. Approach

We divide the approach into three phases. In the first phase, we construct a benchmark which consists of abundant projects with multiple release versions. After that, we obtain the risk profiles in the benchmark and normalize the metric values of each code file to ordinal ratings in a certain range. The risk profile denotes the percentage of overall code that falls into each of the four risk categories: Low, Moderate, High and Very-High. Concretely, the four risk categories are determined by four intervals which are based on the distribution of a metric. Many authors have shown that the distributions of software metrics are heavily skewed [10], [11], [12], thus a typical approach to obtain the intervals is to adopt the threshold set which represents the values at the quantile <70%,10%,10%,10%> [11], [13].

In the second phase, we adopt a topic modeling technique to estimate the probabilistic correlations between quality characteristic, metric and code file. In particular, we adopt a topic modeling technique DPLSA (an extension of PLSA), which assigns a concrete meaning to a topic by a special initialization method [5], [14]. The DPLSA model estimate two probabilistic correlations by learning from the benchmark, namely the probabilistic correlation between metrics and sub-characteristics $P(w \mid z)$ and the probabilistic correlation

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.139

between sub-characteristic and code file $P(z \mid d)$. The two probabilistic correlations form the base of our approach.

In the third phase, we construct a "badness" function for each metric of a system. The outcome represents the badness of the quality characteristic in a system. In detail, we construct badness function at three levels. The first level is the badness of a metric w_j in the system. Let $B(w_j)$ represents the badness function of metric w_j. The method of this level is derived from the work of [10] which aggregates the individual metric value of each class to the whole rating of a system. The second level is the badness of a quality sub-characteristic (e.g., changeability is a sub-characteristic of maintainability) in a system. It is calculated by using the badness of each metric w_j in the first level and the learned probability correlation $P(w \mid z_k)$ between the target sub-characteristic z_k and each metric w by DPLSA. The third level is the badness of a quality characteristic (e.g., maintainability). The badness of the quality characteristic in a system is correlated with the badness of its children nodes (i.e., sub-characteristic) and the probability weight between the code file and the children nodes is also an impacting factor. With the three levels of the badness function, we can assess the high-level quality characteristic of a system.

III. Application

In this section, we attempt to apply the approach for assessing a typical high-level characteristic, namely maintainability on the Qualitas Corpus [15] dataset. The class file is regarded as the document, the sub-characteristics is regarded as topics and the metrics are normalized as words. The goal is to infer the probabilistic correspondence $P(w \mid z)$ and $P(z \mid d)$. Similar to the existing maintainability model [13], [16], we adopt a three-layer quality framework including the target characteristic maintainability, sub-characteristics (i.e., changeability, analyzability, testability, stability as ISO 9126 defined) and metrics. Considering the metrics, we adopt the typical metrics which are identical with the relevant studies of maintainability evaluation [17], [18]. In detail, five Chidambar and Kemerer metrics: WMC, DIT, NOC, RFC, and LCOM; four Li and Henry metrics [19]: MPC, DAC, NOM, and SIZE2; and one traditional lines of code metric (SIZE1) are adopted. SIZE1 represents the number of lines of code excluding comments, and SIZE2 represents the total count of the number of data attributes and local methods in a class.

The result of the application is the badness value of the maintainability. In order to validate the effectiveness of our approach, we adopt a proxy measure of maintainability to evaluate the consistency with the badness value. In this work, we adopt the number of lines of changed code to measure maintainability by following Elish's work [17]. We assume that the badness value has a consistency with the changed lines of code (i.e., the bigger of the badness value, the worse of the maintainability, and the more lines of code need to be changed). In particular, we adopt the Spearman rank-correlation coefficient to evaluate the consistency. The main advantage of the Spearman metric is that it does not require the variables to meet a particular distribution.

TABLE I
The Spearman Coefficient ρ with p-value (in Brackets) in Different Cases.(*Represents Medium Correlation Size; ** Represents Large Correlation Size)

Project	#of versions	Our method	AWLE
Ant	22	0.406*(0.041)	0.500(0.018)
ArgoUml	15	**0.693**(0.005)**	0.527(0.043)
FreeCol	31	**0.438*(0.014)**	0.414(0.020)
Hibernate	51	**0.581**(0.000)**	0.523(0.000)
Jung	22	**0.674**(0.001)**	0.616(0.002)
Antlr	21	0.526**(0.016)	0.5341(0.013)
Azureus	32	**0.349*(0.049)**	-0.359(0.044)
FreeMind	15	**0.607**(0.019)**	0.221(0.429)
JGraph	37	0.223(0.184)	0.132(0.434)
JUnit	23	**0.639**(0.001)**	0.443(0.034)

IV. Results and Analysis

In this paper, we are interesting to answer the research question: **How effective is our model compared with previous method?** Table I shows the correlation analysis results from the benchmark. The Spearman correlation coefficient and the confidence level p-value are provided. For each project, the correlation coefficient represents the consistency level between the badness value and the lines of changed code by considering all the versions. To indicate the correlation size, we follow Cohen's guideline that the correlation coefficient ρ = 0.1, 0.3, and 0.5 represent having small, medium and large correlation sizes [20]. There is only one case whose correlation coefficient is smaller than 0.3. In this case, there is not a significant correlation between maintainability badness and changed lines. However, in the remaining cases, the correlations have a medium or large correlation size and they are significant with at least 95% confidence (p-value $<$ 0.05) level.

In addition, we implement a typical average weighted linear equation (AWLE) method which is adopted in the SIG maintainability model [13] as a baseline. It assigns each metric and sub-characteristic with equal weight. In Table I, the better correlation between the two methods are in bold. The results show that our approach reveal the maintainability better than the AWLE model in most of the projects. Overall, the result suggests that the achieved badness value by using the proposed aggregation method can reveal the maintainability well.

V. Conclusion

In this paper, we propose an aggregation method which attempt to assess the high-level quality characteristic automatically. In detail, we adopted a topic modeling technique to learn the probabilistic correlation between code files, quality characteristics and metrics from a benchmark which consists of numerous release versions of open source projects. With this method, it enables the assessment automatically and overcomes the difficulty in determining the weight in previous weighted linear methods. To evaluate the effectiveness, we conduct an application on the maintainability assessment by using a lot of release versions of open source projects. The results show that our approach can reveal the maintainability well and it performs better than a previous baseline through a correlation analysis with the changed lines of code.

REFERENCES

[1] S. Wagner, A. Goeb, L. Heinemann, M. Kls, C. Lampasona, K. Lochmann, A. Mayr, R. Plsch, A. Seidl, J. Streit, and A. Trendowicz, "Operationalised product quality models and assessment: The quamoco approach," *IST*, vol. 62, pp. 101–123, 2015.

[2] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *TSE*, vol. 28, no. 1, pp. 4–17, 2002.

[3] M. Morisio, I. Stamelos, and A. Tsoukias, "Software product and process assessment through profile-based evaluation," *IJSEKE*, 2003.

[4] T. Hofmann, "Unsupervised learning by probabilistic latent semantic analysis," *Machine Learning*, vol. 42, no. 1, pp. 177–196, 2001.

[5] M. Yan, Y. Fu, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "Automatically classifying software changes via discriminative topic model: Supporting multi-category and cross-project," *JSS*, 2016.

[6] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *TSE*, 2016.

[7] Y. Fu, M. Yan, X. Zhang, L. Xu, D. Yang, and J. D. Kymer, "Automated classification of software change messages by semi-supervised latent dirichlet allocation," *IST*, vol. 57, pp. 369–377, 2015.

[8] J. Zou, L. Xu, W. Guo, M. Yan, D. Yang, and X. Zhang, "Which non-functional requirements do developers focus on? an empirical study on stack overflow using topic analysis," in *MSR*. IEEE, 2015, pp. 446–449.

[9] X. Xia, D. Lo, X. Wang, and B. Zhou, "Dual analysis for recommending developers to resolve bugs," *JSEP*, vol. 27, no. 3, pp. 195–220, 2015.

[10] T. L. Alves, J. P. Correia, and J. Visser, "Benchmark-based aggregation of metrics to ratings," in *IWSM-MENSURA*, 2011, pp. 20–29.

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

[12] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *TSE*, 2007.

[13] R. Baggen, J. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Quality Journal*, vol. 20, no. 2, pp. 287–307, 2012.

[14] M. Yan, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "A component recommender for bug reports using discriminative probability latent semantic analysis," *IST*, vol. 73, pp. 37–51, 2016.

[15] E. Tempero, C. Anslow, J. Dietrich, T. Han, L. Jing, 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.

[16] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy, "A probabilistic software quality model," in *ICSM*, 2011, pp. 243–252.

[17] M. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on predicting software maintainability using ensemble methods," *Soft Computing*, vol. 19, no. 9, pp. 2511–2524, 2015.

[18] C. van Koten and A. R. Gray, "An application of bayesian network for predicting object-oriented software maintainability," *IST*, 2006.

[19] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *JSS*, vol. 23, no. 2, pp. 111–122, 1993.

[20] D. Athanasiou, A. Nugroho, J. Visser, and A. Zaidman, "Test code quality and its relation to issue handling performance," *TSE*, 2014.