Automating Aggregation for Software Quality Modeling

Meng Yan*‡, Xin Xia†✓, Xiaohong Zhang‡✓, Dan Yang‡ and Ling Xu‡
*College of Computer Science and Technology, Zhejiang University, Hangzhou, China
†Department of Computer Science, University of British Columbia, Canada
‡School of Software Engineering, Chongqing University, Chongqing, China
Email: mengy@zju.edu.cn, xxia02@cs.ubc.ca, {xhongz, dyang, xuling}@cqu.edu.cn

***Abstract*—Software Quality model is a well-accepted way for assessing high-level quality characteristics (e.g., maintainability) by aggregation from low-level metrics. Aggregation method in a software quality model denotes how to aggregate low-level metrics to high-level quality characteristics. Most of the existing quality models adopt the weighted linear aggregation method. The main drawback of weighted linear method is that it 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 leverage 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 probabilistic relationship without manual effort.**

To evaluate the effectiveness of proposed aggregation approach, we conduct an empirical study on assessing one typical high-level quality characteristic (i.e., maintainability) which is regarded as an important characteristic defined in ISO 9126. The achieved results on 10 open source projects with totally 269 versions show that our method can reveal maintainability well and it outperforms a weighted linear aggregation method baseline in most of the projects.

***Keywords*-Software Quality Modeling, Aggregation Method, Topic Model**

I. Introduction

It is crucial for both developers and managers to understand and assess quality of a software product. Although the quality standards (e.g., ISO 9126 and ISO 25010) provide the definition for the quality characteristics, they do not provide a detailed approach to measure them. A software quality model aims to provide the specific assessment results about different high-level quality characteristics (e.g., functionality, reliability, usability, efficiency, maintainability and portability defined in ISO 9126) by aggregation from low-level software metrics (e.g., number of source lines of code) [1]. This quality assessment result is mainly used for quality assurance, decision making, costs estimating and risk evaluation during software development and maintenance.

The aggregation method is an important part of any quality model and a reoccurring task in any measurement method. However, decisions of the aggregation method for software quality model are rarely justified in this line of research. The most commonly 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 classification [5]–[8]. 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 as presented in Figure 1. 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.

To evaluate the effectiveness of proposed method, we target on one typical high-level quality characteristic, namely Maintainability by following the work of Bakota et al. [9]. The reason for choosing maintainability is that it is regarded as an important characteristic defined in ISO 9126 owing to its

✓Corresponding authors.

978-1-5386-0992-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSME.2017.30

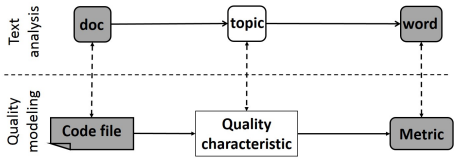


Fig. 1. Similarity between text analysis and quality modeling. The filled entry represents the observed layer while the blank entry represents the hidden layer.

direct impact on development and maintenance costs [9]. In summary, the main contributions of this paper are as follows:

- We propose an automated aggregation approach for assessing high-level software quality characteristics. It enables automated quality modeling by using the leaned probabilistic correlation without manual effort.
- In order to evaluate the effectiveness of proposed method, we conduct an empirical study for maintainability modeling on 10 open source projects with totally 269 versions. The achieved results show that our method can reveal maintainability well and it outperforms a baseline in most of the projects.

II. Related Work

Along with the widely accepted standards (e.g., ISO/IEC 9126), a multitude of diverse models for assessing software quality have been proposed [10], such as QMOOD [2], Squale [11], SQALE [12] and SIG model [13]. One common issue in the above-mentioned models is that the aggregation step from metrics towards characteristics is conducted by weighted linear equations (WLE). The weights in these works are determined by experts opinions which may make the aggregation subjective due to lack of consensus among experts [8]. There are other studies which adopt new methods in the aggregation step, such as fuzzy logic based approach [14], outrank relation [3], [15], and geometric mean [16]. One issue in these works is that they are suitable for a particular semantic context. For example, the geometric mean method is suitable for requiring all the child nodes to have a high score which may not suitable for other conditions. Besides, Bakota et al. [9] proposed a probability quality model which emphasized the probabilistic distribution in the aggregation step. However, a weight of each node in their model is still required which is determined by a manual survey in their work. It is difficult for reoccurring their work due to the manual effort. While in our work, we adopt the weight which is automatically learned from benchmark to replace the manual effort.

III. Approach

We divide the approach into three phases as Figure 2 shows. 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. In the second phase, we adopt a topic modeling technique to estimate the probabilistic correlations between quality characteristic, metric and code file. 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.

A. Benchmark Normalization Phase

This section describes the process of benchmark normalization. The goal of normalization is to transform the different metric values with different ranges into a same scale. For simplification, we divide the process into two steps. The first step is to build risk profiles according to the benchmark thresholds. 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 [17]–[19], thus a typical approach to obtain the intervals is to adopt the threshold set which represents the values at the quantile $<$70%,10%,10%,10%$>$ [13], [18].

After deciding the thresholds, the second step is to build the co-occurrence table by transforming the risk level of each code file into an ordinal number. Formally, let d_i denotes the i-th code file in the benchmark, w_j denotes the j-th metric, and $n(d_i, w_j)$ denotes the risk level of the w_j in d_i. Similar to text mining, the input of the topic modeling should be a co-occurrence table which consists of integers. Thus, we transform $n(d_i, w_j)$ into 1, 2, 3, 4 which correspond to the intervals $<$70%, 10%, 10%, 10%$>$ respectively. As a result, the numbers 1, 2, 3, 4 denote Low, Moderate, High and Very-High risk respectively.

B. Topic Modeling Phase

This section presents how to learn the probabilistic correlation between quality characteristic, metric and code file. We adopt a topic modeling technique DPLSA (an extension of PLSA), which assigns a concrete meaning to a topic by a special initialization method [20], [21]. The input of DPLSA model is the co-occurrence table which is derived from previous subsection and an initial weight table. In text analysis, one feature of the DPSLA model is that the topic has a one-to-one correspondence with the categories of words due to the initial table. Similarly, since the metrics are directly connected with sub-characteristics, the topic has a one-to-one correspondence with the sub-characteristic by using DPLSA. The initialization table provides an initialized connection between the metrics and the sub-characteristics. It derives from prior knowledge which represents the connections of a metric and a sub-characteristic. Once the initialization table is provided, the DPLSA model estimates two probabilistic correlations by learning from the benchmark, namely the probabilistic correlation between metrics and sub-characteristics and the probabilistic correlation between sub-characteristic and code file. The two probabilistic correlations form the base of our approach.

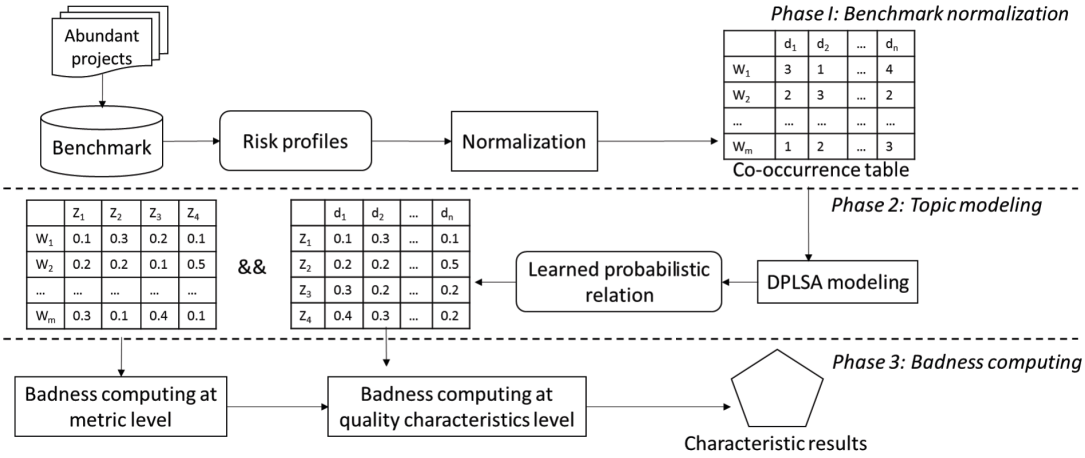


Fig. 2. Overview of proposed approach

C. Badness Calculation Phase

This section presents how to perform aggregation by constructing badness functions at the following three levels.

Level 1. The first level is the badness of a metric w_j in the system. Let $B(w_j)$ represent the badness function of metric w_j. The calculation method of this level is derived from the work of Alves et al. [17] which aggregates the individual metric value of each class to the whole rating of a system.

Level 2. The second level is the badness of a quality sub-characteristic (e.g., changeability is a sub-characteristic of maintainability as ISO 9126 defined) in a system. Earlier, we have obtained two variables which are correlated with the badness of a quality sub-characteristic: 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. Thus, suppose there are N metrics in total, we construct the badness function of the quality sub-characteristic $B(z_k)$ as follows:

$$B(z_k) = \sum_{j=1}^{N} B(w_j) P(w_j \mid z_k) \tag{1}$$

Level 3. 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). In addition, the probability weight between the code file and the children nodes is also an impacting factor. Thus, we construct the badness function of the quality characteristic $B(Q)$ of a system as follows:

$$B(Q) = \sum_{k=1}^{K} B(z_k) P(z_k \mid s) \tag{2}$$

In (2), $P(z_k \mid s)$ represents the probability weight between the system and the topic z_k. However, we only learn the $P(z_k \mid d_i)$ which is not a system-level representation, it is the probability correlation between the topic z_k and code file d_i in the system s. Since the correlation between a system and a topic is determined by all of the code files, we calculate the system-level $P(z_k \mid s)$ by using the average $P(z_k \mid d_i)$ of all its code files. With the three levels of the badness function, we can assess the high-level quality characteristic of a system.

IV. Experimental Setup

In this section, we attempt to apply the approach for assessing a typical high-level characteristic, namely maintainability, which is regarded as an important characteristic defined in ISO 9126 [9].

A. Maintainability Model Description

Similar to the existing maintainability models [9], [13], 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 [22], [23]. In detail, five Chidamber and Kemerer metrics: WMC, DIT, NOC, RFC, and LCOM; four Li and Henry metrics [24]: 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.

Note that although the proposed approach is independent with metrics and characteristics, the prior knowledge which represents the connection between characteristics and metrics is required to determine the initialization step in DPLSA. In this application, the prior knowledge contains two aspects. The first is that the maintainability is correlated with four sub-characteristics, changeability, analyzability, testability and stability. The second is that each sub-characteristic is correlated with all of the selected 10 metrics (i.e., a fully connected framework) as the work of [22], [23] stated. The projection on the topic model of this quality framework is visualized as presented in Figure 3. In detail, the class file is regarded as the document, the sub-characteristics have a one-to-one correspondence with topics and the metrics are normalized as words. The goal is to infer the probabilistic correspondence $P(w \mid z)$ and $P(z \mid d)$.

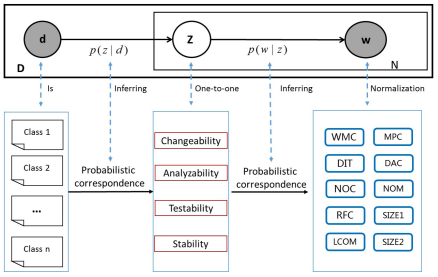


Fig. 3. The projection on topic model of the maintainability model

B. Dataset

Currently, we adopt the public dataset Qualitas Corpus [25] (Qualitas Corpus version is 20130901e) to construct the benchmark. All the systems in the benchmark are written in Java programming language and have multiple evolution versions. For each system, the distributions of metrics over each code file (i.e., class) and the normalized values are stored in our benchmark.

C. Performance Measure

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 past studies, the maintainability can be measured in many ways, such as the time required to make changes or the number of lines of changed code. In this work, we adopt the number of lines of changed code to measure maintainability by following Elish's work [22]. 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). Then, we adopt the Spearman rank-correlation coefficient to evaluate the consistency.

V. Experimental Results and Analysis

In this section, we present empirical results of the following two research questions.

RQ1: How effective is our approach for assessing software maintainability?

RQ2: What insights can we observe from the maintainability evolution of sequential release versions in a project?

A. RQ1: Effective Analysis

Motivation. Our objective is to provide an automated aggregation method for software quality modeling. The first question for using the proposed approach is to evaluate how effective it is for quality modeling. Since a holistic quality modeling is a complex task, we attempt to evaluate our approach for maintainability modeling [9].

Results. Table I shows the correlation analysis results of 10 projects randomly selected from the benchmark. The Spearman correlation coefficient and the confidence level p-value are listed. For each project, the correlation coefficient represents the consistency degree between the badness value and the lines of changed code by considering all the versions.

TABLE I
The Spearman Coefficient 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)

To indicate the correlation size, we follow Cohen's guideline that the correlation coefficient = 0.1, 0.3, and 0.5 represent having small, medium and large correlation sizes [16]. There is only one case whose correlation coefficient is smaller than 0.3. In this case, there is no 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.

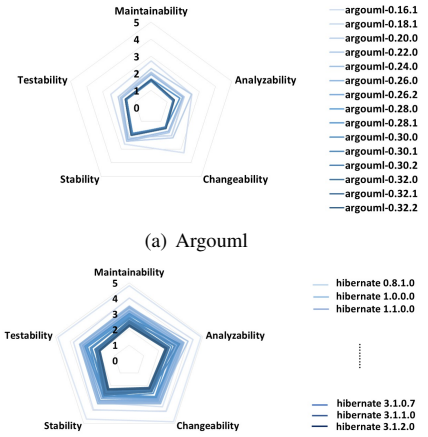
B. RQ2: Evolution Analysis

Motivation. The output of our approach is a maintainability badness value which can provide an overall assessment of a project version. In our experiment, each project consists of many sequential versions. We are interested in how does the maintainability evolve along with the release versions. The expected usage scenario is: if the upcoming release version has an unusual maintainability which does not match the previous evolution pattern, it is time to conduct more software quality assurance activities, such as code review or refactoring.

Results. Figure 4 presents the badness evolution with the release versions of two projects (Argouml and Hibernate) randomly selected in our dataset. There are 15 versions in Argouml and 51 versions in Hibernate. Overall, our approach reveals that the badness of maintainability and its four sub-characteristics become smaller along with the release versions. It suggests that there is an overall trend in the two projects: maintainability is better along with the version evolution in the two projects. In summary, our approach can provide a overall trend of maintainability evolution along with release versions.

VI. Threats to Validity

Threats to internal validity relates to the setting in our experiments. First, we adopt a widely used proxy measurement (i.e., changed lines of code) as the ground truth selection of maintainability in our experiments. This might be a threat since the maintainability effort may be impacted by other



(a) Argouml

(b) Hibernate

Fig. 4. The maintainability badness evolution of two projects (15 versions of Argouml and 51 versions of Hibernate)

factors (e.g., time elapsed between two versions). Second, the baseline selection in the comparison of the experiments might be bias. We adopt a typical method AWLE as the baseline, since it is an automated method without the need of manual effort. This might be a threat since we do not consider other aggregation methods. A more refined work is needed by comparing with more aggregation methods. Third, we adopt the threshold set which represents the values at the quantile $<$70%,10%,10%,10%$>$. This might be a threat since the distribution of different metrics may differ in their types.

Threats to external validity relates to the generalization of our approach. We have evaluated our aggregation method on the maintainability model by conducting the experiments on Java open source projects. In the future, we plan to mitigate this threat further by evaluating our method on more kinds of quality model and software systems.

VII. Conclusion

In this paper, we propose an aggregation method for automating software quality modeling. We leverage 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. By using the learned probabilistic correlation, our approach enables quality modeling automatically and overcomes the difficulty in determining weight in previous weighted linear methods. To evaluate the effectiveness of proposed approach, we conduct an empirical study for one typical software quality characteristic (i.e., maintainability) modeling by using a lot of release versions of open source projects. The experimental results show that our approach can reveal maintainability well and it performs better in most of the projects than a previous weighted linear method baseline. In addition, our approach can reveal the overall trend of maintainability evolution considering sequential release versions in a project. This evolution trend can be used to identify release version with unusual low maintainability.

Acknowledgment. This work was partially supported by NSFC Program (No. 61602403 and 61572426), Chongqing Research Program of Basic Science & Frontier Technology (No. cstc2017jcyjB0305).

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] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] M. Klas, J. Heidrich, J. Munch, and A. Trendowicz, "Cqml scheme: A classification scheme for comprehensive quality model landscapes," in *SEAA*, 2009, pp. 243–250.

[11] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, and P. Vaillergues, "The squale model-a practice-based industrial quality model," in *ICSM*, 2009, pp. 531–534.

[12] J.-L. Letouzey and T. Coq, "The sqale analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code," in *VALID*, 2010.

[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] J. S. Challa, A. Paul, Y. Dada, V. Nerella, P. R. Srivastava, and A. P. Singh, "Integrated software quality evaluation: A fuzzy multi-criteria approach," *JIPS*, vol. 7, no. 3, pp. 473–518, 2011.

[15] I. Samoladas, G. Gousios, D. Spinellis, and I. Stamelos, *The SQO-OSS Quality Model: Measurement Based Open Source Software Evaluation*, ser. IFIP, 2008, vol. 275, book section 19, pp. 237–248.

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

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

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

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

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.