Quality Assessment for Large-Scale Industrial Software Systems: Experience Report at Alibaba

Chen Zhi*†, Shuiguang Deng*†, Jianwei Yin*, Min Fu‡§, Hai Zhu‡, Yuanping Li‡, Tao Xie¶
*Zhejiang University, Hangzhou, China
†Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Hangzhou, China
‡Alibaba Group, Hangzhou, China
§Macquarie University, Sydney, Australia
¶Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, China
{zjuzhichen, dengsg, zjuyjw}@zju.edu.cn, {hanhao.fm, marvin.zh, yuanping.lyp}@alibaba-inc.com, taoxie@pku.edu.cn

***Abstract*—To assure high software quality for large-scale industrial software systems, traditional approaches of software quality assurance, such as software testing and performance engineering, have been widely used within Alibaba, the world's largest retailer, and one of the largest Internet companies in the world. However, there still exists a high demand for software quality assessment to achieve high sustainability of business growth and engineering culture in Alibaba. To address this issue, we develop an industrial solution for software quality assessment by following the GQM paradigm in an industrial setting. Moreover, we integrate multiple assessment methods into our solution, ranging from metric selection to rating aggregation. Our solution has been implemented, deployed, and adopted at Alibaba: (1) used by Alibaba's Business Platform Unit to continually monitor the quality for 60+ core software systems; (2) used by Alibaba's R&D Efficiency Unit to support group-wide quality-aware code search and automatic code inspection. This paper presents our proposed industrial solution, including its techniques and industrial adoption, along with the lessons learned during the development and deployment of our solution.**

***Keywords*—Software quality assessment, software quality model, experience report**

I. Introduction

With the widespread application of computing technologies, various industries have become increasingly dependent on software systems. At the same time, due to various software quality problems, ranging from poor user experiences to software failures, more and more severe accidents arise, not only resulting in substantial economic losses [1], but also sometimes even human-life losses [2]. Due to the importance of software quality, significant efforts have been invested by both academia and industry to assure high software quality. As a result, a large number of software quality assurance technologies have been proposed and applied, such as software measurement, software testing, and code review.

Alibaba is the world's largest retailer, and one of the largest Internet companies in the world. It owns and operates a diverse array of businesses worldwide, such as e-commerce services, electronic payment services, and cloud computing services. In order to support such a large-scale business, the size of the underlying software systems is substantial, amounting to several billion lines of code. The size is increasing continuously with the rapid growth of the business. Significant efforts have been made to assure high software quality for such large-scale software systems. For example, we have built substantial testing infrastructures (e.g., the automatic regression testing platform and end-to-end stress testing platform) to assure correct functionality and high reliability. Moreover, typically the foundational system software (e.g., JVM and Tomcat) is highly customized and optimized to improve the performance.

However, there still exists a high demand for software quality assessment in Alibaba for two main reasons:

- **Ensuring high sustainability of business growth.** As the rapid growth of Alibaba, it has become more than just an e-commerce platform, but a vast commerce ecosystem across many industries. The services provided by Alibaba have become the infrastructure for the business industry as well as an essential part of Chinese daily life. As the businesses of Alibaba highly depend on the underlying software systems, they should be easy to maintain and evolve to support the sustainable business growth of Alibaba.
- **Building engineering culture.** In order to develop core technologies to support business innovation, Alibaba is transforming from business-driven to technology-driven. Building engineering culture is a critical part of the transformation, as doing so can inspire the creativity of engineers. Software quality assessment is perceived as a vital step toward engineering culture: higher-quality software typically requires less time to maintain and allows developers to have more time to do creative work.

As a supplement to existing practices at Alibaba, we aim to develop an industrial solution for software quality assessment similar to a physical medical examination. First, there is a rating representing the overall software quality (similar to the health condition) so that the developers and managers can obtain a direct understanding of the quality condition. Second, there are various attributes for reflecting different aspects of the software (similar to human body), and the overall rating is aggregated from ratings for these attributes. Each attribute can be used to identify problems (similar to health symptoms), which can help pinpoint problematic code (similar to disease). Third, quality assessment is fully automatic without human

2640-0715/19/$31.00 ©2019 IEEE
DOI 10.1109/APSEC48747.2019.00028

intervention, because the efficiency and scalability of manual assessment are infeasible when dealing with large-scale industrial software systems.

To produce such an industrial solution, we conduct a systematic review to understand existing techniques for software quality assessment. After that, we adapt and improve existing techniques to build our solution for software quality assessment. Our industrial solution has been implemented, deployed, and adopted at two major units at Alibaba. First, it is adopted by Alibaba's Business Platform Unit, which is responsible for developing foundational platforms to provide basic business capability (e.g., membership management, trade management, and fund management) and support a number of business lines (e.g., Taobao, TMall, and AliExpress). Our solution has been continually monitoring the quality for 60+ core software systems and has operated stably for about one year. Second, it is adopted by Alibaba's R&D Efficiency Unit, which is in charge of developing supporting platforms to improve the efficiency and quality of software development in Alibaba. Our solution is adopted to support quality-aware code search and automatic code inspection.

In this paper, we present our proposed industrial solution for software quality assessment developed upon adapting and improving existing techniques. Furthermore, we summarize the lessons that we learn to provide useful guidelines for other researchers or practitioners to conduct technology transfer of software quality assessment to industrial practices.

In summary, this paper makes the following main contributions:

- A brief overview of the general process and existing techniques for software quality assessment. Moreover, we give some brief explanation for the choices of technology genres.
- A detailed description of our proposed industrial solution, including its techniques and industrial adoption.
- Lessons learned during the development and deployment of our proposed industrial solution, from both technical aspects and social aspects.

The rest of the paper is organized as follows. Section II discusses the background and related work. Section III presents the technical details of our proposed industrial solution. Section IV describes our industrial adoption at Alibaba. Section V discusses the lessons learned, and Section III concludes and outlines future work.

II. Background and Related Work

To build a system for software quality assessment, there are generally two phases: building a model of software quality and defining a method of software quality assessment. Since software quality is a complex and multifaceted concept, a quality model is a common way to define software quality in a structured manner. Ideally, a quality model can decompose the quality into some software metrics that represent certain aspects of quality. However, the measurement results usually cannot be directly used for quality assessment for two reasons. First, there is often a gap between software metrics and software quality, so we need to transform the raw metric values to ratings that represent software quality. Second, many metrics are defined at low-level entities, such as methods and classes, and aggregation is required to derive a single value for the entire project under assessment. Therefore, the assessment method should clarify both how to rate and how to aggregate.

A. Model of Software Quality

A large number of quality models have been proposed since the emergence of this research topic in the 1970s. At the early stage, models such as Boehm [3] and McCall [4] explore various characteristics of software quality in a top-down fashion. Based on these early quality models, the ISO/IEC 9126 standard was defined in 1991, and its successor ISO/IEC 25010 was proposed in 2011. These standards provide reference models for the quality of software products. However, these standards are often too complex to be practical. Therefore, some recent work [5]–[8] focuses on adapting these standard models in practice. Compared with such work, our work focuses on building a model of software quality from software quality problems observed in practice by following the Goal-Question-Metric (GQM) paradigm [9], especially providing some guidelines for detailed procedures (e.g., metric selection and threshold derivation). Additionally, the model derived from our work can be integrated with standard quality models by mapping software quality problems observed in practice to quality factors.

B. Method of Software Quality Assessment

A method of software quality assessment consists of two parts: rating and aggregation. We next provide a brief overview of existing techniques on these two parts.

Existing techniques of software quality rating mainly fall into three categories: threshold-based techniques [10], utility-function-based techniques [7], and probabilistic techniques [11]:

- **Threshold-based techniques** [10] assign an approximate risk level or profile to specific metric values according to predefined threshold values. However, the number of levels is finite, indicating that many different metric values can fall into the same level, and the expressiveness of the assessment result is limited.
- **Utility-function-based techniques** [7] can be viewed as continuous generalizations of threshold-based techniques. There are two steps for utility-function-based techniques: (1) define the worst and best cases for the metrics and assign the lowest and highest scores to the corresponding metric values, and (2) define a utility function that can fit these two fixed points. However, it is difficult to determine the worst or best case for some metrics. For example, as the metric of Lines Of Code in method (LOCm) follows a power-law distribution, any extreme value is possible to appear.
- **Probabilistic techniques** [11] are the most desirable techniques, because the assessment result is not a single

value, but a probability distribution. The key of probabilistic techniques is constructing the goodness function (the probabilistic generalizations of the utility function), from histograms of metric values. However, the assessment result of this technique is hard to understand.

In summary, the three categories of techniques increase the expressiveness in order, but the explainability decreases. In our work, we adapt the first two categories of techniques because having a concrete value to represent software quality is one of the requirements for our solution. We choose the most suitable rating techniques according to the characteristics of software metrics, such as value ranges and data distributions.

Since we use threshold-based techniques to rate some software metrics, we need to determine the threshold values for each software metric. There are generally two types of techniques to derive thresholds: expert-based techniques and data-driven techniques. The former relies on experts as they can define the thresholds based on their experiences. However, there are many software metrics lacking off-the-shelf thresholds defined by experts. The latter usually collects metric data for a number of benchmarking software systems and derive some percentiles of the metric-data distribution as the thresholds. However, it is challenging to determine reasonable percentiles. To address these issues, we combine the two types of techniques by transferring expert knowledge based on data distributions.

Many techniques have been applied to aggregate software metrics [12], ranging from summation, measuring accumulative effect, to some advanced concepts such as inequality indices, measuring the degree of imbalance in distribution. We use multiple aggregation techniques, including simple average, weighted average, and logarithm-based aggregation [13], and we choose the most suitable ones according to the requirements of aggregation, such as highlighting problematic code.

III. Our Industrial Solution

In this section, we describe how we adapt and improve existing techniques for software quality assessment and give the details of our industrial solution, including building a model of software quality and defining a method of software quality assessment, along with the implementation and industrial adoption.

A. Building a Model of Software Quality

We leverage the GQM paradigm [9] to build a model of software quality. This paradigm generally consists of three steps: (1) determining the goal of the stakeholders, (2) defining the question that must be answered to determine whether the goal is being met, and (3) deciding what must be measured to answer the question accurately.

1) Determining Goal: As the first step, we follow the GQM template and formulate our goal as below:

> *"Analyze **source code** for the purpose of **quality assessment** with respect to **maintainability** from the point of view of **developers** in the context of **large-scale industrial software systems**."*

We choose to analyze the source code, but not the process or other products, because we intend to estimate the quality risk at the early stage of the software development.

2) Defining Question: For the second step, we determine the question by collecting the requirements from developers for the industrial software systems. In this manner, our quality model can reflect what the developers' concerns are. In particular, we conduct an informal interview with 20 technical staff members at Alibaba to ask them about their opinions on software quality and what quality factors should be included for software quality assessment. The participants are from 4 different business units of Alibaba and consist of 12 senior developers, 4 chief architects, and 4 senior testers. Each of them has more than 10-year development experience. From the result of the informal interview, we identify four most commonly raised questions (corresponding to four dimensions of software quality), as listed below:

- **Coding convention**. How well does the code comply with Alibaba Java Coding Guidelines [14]? The guidelines consolidate the best programming practices from Alibaba Group's technical teams. The guidelines can help developers minimize potential and repetitive coding mistakes.
- **Code duplication**. How much code duplication does the project include? Code duplication can increase the potential risk and maintenance effort, as we need to ensure consistent changes to duplicated code fragments.
- **Complexity**. How complex is a code unit in the project? A code unit (e.g., method and class) is the lowest-level piece of functionality. It should be kept of low complexity to assure high understandability and maintainability.
- **Object-oriented (OO) design**. How well does the code conform to the object-oriented paradigm? As most backend systems of Alibaba are written in Java, and the OO design quality can reflect architecture quality to some extent, which also is an integral part of software quality.

3) Deciding Metrics: The final step is metric selection. We select metrics according to two criteria: **extensive validation** and **wide acceptance**. Meneely et al. [15] pointed out that predictability is one of the most widely used criteria for software metric validation. The quality risk that we intend to estimate can be represented by the number of post-release defects. Therefore, we select those metrics that are highly predictable for post-release defects.

However, few past research efforts use coding-convention metrics and code-duplication metrics to predict post-release defects. So we use only two well-accepted metrics in practice: density of rule violations denoted as $density(r\_v)$ and duplication coverage denoted as $cov(dup)$, respectively. The former is defined as the number of rule violations divided by the total number of lines of code from the project under measurement. The latter is defined as the total number of lines of duplicate methods divided by the total number of lines of code from the project. A method m is marked as duplicate if there is another method whose similarity with m exceeds the predefined threshold.

TABLE I: OVERVIEW OF OUR INDUSTRIAL SOLUTION

Dimension	Metric	Interpretation	Rating	Threshold	Aggregation		
					Across Entities	Across Metrics	Across Dimension
Coding Convention	density(r_v)	Density of rule violations	Max function based	N/A	N/A	N/A	Weighted Average
Code Duplication	cov(dup)	Duplication coverage	Linear function based	N/A	N/A	N/A	
Complexity	CC	Cyclomatic complexity	Threshold based	[10, 20, 35, 50]*	Logarithm-based	Simple Average	
	LOCm	Lines of code in method	Threshold based	[30, 45, 70, 100]*	Logarithm-based		
	MND	Max nested level	Threshold based	[4, 5, 6, 7]*	Logarithm-based		
	NOP	Number of parameters	Threshold based	[3, 5, 6, 7]+	Logarithm-based		
Object-oriented Design	NOM	Number of methods	Threshold based	[15, 20, 30, 50]*	Logarithm-based	Simple Average	
	NOF	Number of fields	Threshold based	[5, 7, 11, 20]*	Logarithm-based		
	WMC	Weighted method count	Threshold based	[50, 150, 250, 350]+	Logarithm-based		
	CBO	Coupling between objects	Threshold based	[20, 40, 60, 80]+	Logarithm-based		
	RFC	Response for class	Threshold based	[60, 120, 180, 240]+	Logarithm-based		
	DAM	Data access metric	Linear function based	N/A	Logarithm-based		
	LCOM_HS	Lack of cohesion of method	Linear function based	N/A	Logarithm-based		

* Expert-based Threshold Derivation + Data-driven Threshold Derivation

Fortunately, software metrics measuring complexity and OO design are an active research topic, and a number of metrics have been used to predict maintainability efforts and post-release defects. Based on the result of previous literature reviews [16], [17], we select 11 metrics for complexity and OO design, respectively, as listed in Table I: 1 metric for coding convention, 1 metric for code duplication, 4 metrics for complexity, and 7 metrics for OO design.

B. Defining a Method of Software Quality Assessment

We next present the two parts in our defined method of software quality assessment: rating and aggregation.

1) Rating: We use threshold-based rating and utility-function-based rating in our work. We prefer utility-function-based rating when we can define the worst case and best case for specific metrics.

The rating of coding convention is derived from the density of rule violations denoted as $density(r\_v)$. Although the value range of $density(r\_v)$ is $[0, +\infty]$, we observe that the maximum value does not exceed 1 in real data distributions. Therefore, the best case for $density(r\_v)$ is 0, which indicates that there is no rule violation, and the worst case for $density(r\_v)$ is 1, which indicates that every line has one rule violation on average. We use the max function to rate coding convention, with the following formula:

$$rating(conv) = max((1 - density(r\_v)) * 100, 0) \quad (1)$$

The rating of code duplication is based on duplication coverage denoted as $cov(dup)$. The value range of $cov(dup)$ is $[0, 1]$. The best case for $cov(dup)$ is 0, which indicates that there is no code duplication, and the worst case for $cov(dup)$ is 1, which indicates that every method is duplicate. The formula of duplication rating is

$$rating(dup) = (1 - cov(dup)) * 100 \quad (2)$$

We mainly use threshold-based techniques to rate the 11 complexity metrics and OO design metrics as shown in Table I, because 9 out of these 11 metrics follow the power-law distribution, indicating that it is hard to define the worst case for these metrics. At first, we decide to use the 5-point risk level: very high, high, mediate, low, and very low, as Morisio et al. [10] pointed out that the ideal number of risk levels is between three and five. In order to ease the aggregation among different risk levels, we assign continuous interval scores, from 0 to 4, to these levels, respectively.

However, it is difficult to determine reasonable thresholds for these metrics, even after many attempts have been made by researchers for decades. To overcome these drawbacks, we combine expert-based techniques and data-driven techniques with the following procedure:

1) **Threshold collection**. We perform a literature review and web search to collect existing thresholds for five metrics, including NOM, NOF, CC, LOCm, and MND.
2) **Benchmark construction**. As Klaus et al. [18] pointed out that the larger the size of the benchmarking base is, the less the variance of the quality assessment result is. We construct a benchmark base by retrieving all public and successfully compiled repositories written in Java in an internal GitLab, which is the most widely used version control system in Alibaba. The resulting benchmark consists of more than 5,000 software systems with various sizes and different application domains.
3) **Distribution analysis**. We observe that existing thresholds of these five metrics approximately represent 95%, 99%, 99.5%, and 99.9% quantiles of the corresponding data distribution in the benchmark. Figure 1 shows the distributions and thresholds of NOM and NOF. Due to the large scale of the benchmark, the data distribution is extremely right-skew, and these percentiles are very close to each other.
4) **Threshold transfer**. For the remaining metrics following the power-law distribution, we use the 95%, 99%, 99.5%, and 99.9% quantiles as their thresholds.

The threshold values used in our solution are listed in

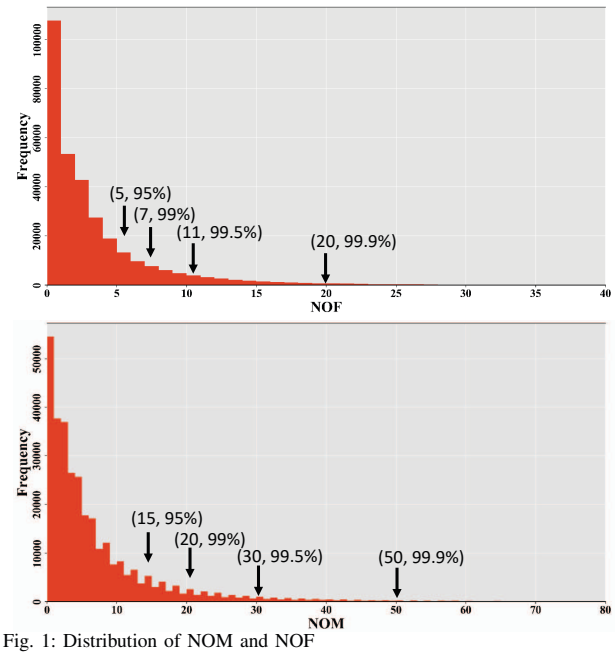

Fig. 1: Distribution of NOM and NOF

Table I. Note that the ratings of DAM and LCOM_HS are calculated by utility-function-based techniques, specifically, linear-function-based rating. Because these two metrics are both ratio-scale-type metrics with closed intervals, it is easy to define the best case and worst case for these two metrics.

2) Aggregation: After we get ratings for each metric, we aggregate all these ratings to one overall rating. The rating aggregation includes three steps. First, we aggregate ratings of low-level entities (such as methods and classes) to get the rating for each metric at the system level. Second, we aggregate ratings of metrics in each dimension (of the four dimensions listed in Section III-A2) to get ratings of dimensions. Finally, we aggregate ratings of dimensions to get the overall rating.

In the first step, most existing work aggregates low-level ratings into top-level ratings directly. However, doing so is unacceptable for quality assessment, because one of the requirements for our industrial solution is to pinpoint problematic code. When developers investigate the identified problematic code, it is more desirable for the developers to drill down step by step to lower-level entities. So we decide to aggregate ratings across the code structure, including methods, classes, files, modules, and systems. As most of the metrics follow the power-law distribution, indicating that the majority of the code has high ratings and low risk. Therefore, we decide to use the logarithm-based aggregation to highlight the problematic code, with the following formula:

$$y = \log_{10} \frac{\sum_{i=1}^{n} 10^{-x_i}}{n} \tag{3}$$

where x_i is the low-level ratings and y is the resulting high-level rating.

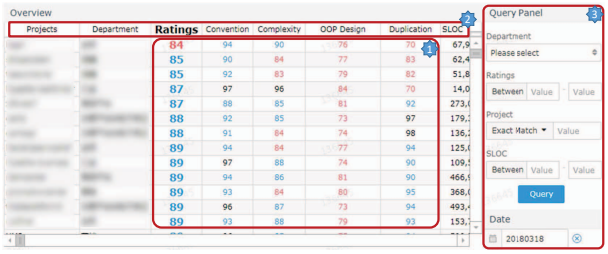

Fig. 2: An Example of Dashboard

To aggregate ratings of different metrics, we use the simple average function to aggregate ratings of metrics to ratings of dimensions. Most existing work uses the weighted-average function for aggregation. However, it is difficult even for experts to determine the weight of metrics. Therefore, we use the simple average function instead.

We use the weighted-average function to aggregate ratings of dimensions to the overall rating. Because these four dimensions are extracted from our interview with industrial developers, we conduct another survey to ask about 50 industrial developers to assign a weight for each dimension. One third of the participants each have more than 10-year development experiences, and the rest of them each have at least 5-year development experiences. We use the average weight as the final weight for each dimension, being 0.34 (coding convention), 0.25 (code duplication), 0.215 (complexity), and 0.195 (OO design).

IV. Industrial Adoption of Our Solution

A. Adoption by Alibaba's Business Platform Unit for core software development

Alibaba's Business Platform Unit is responsible for developing foundational platforms, such as membership management, trade management, and fund management, to support various business lines, such as Taobao, TMall, and AliExpress. Thus, the quality of these foundational platforms is critically important to the Alibaba business.

Based on our solution, we develop a prototype to measure and analyze the quality of these foundational platforms. The prototype pulls the latest code from version control systems (e.g., Git and Subversion), collects raw metric data by state-of-the-art tools (e.g., *p3c-pmd* [19] for coding convention and *SourcererCC* [20] for clone detection), stores rating data into a large-scale data warehousing system [21], and visualizes rating data by powerful analysis tools [22]. Figure 2 shows an example of a dashboard for some projects. All the ratings are labeled in different colors based on the values to highlight low ratings (Figure 2-1). Each column can be sorted as the descending or ascending order to support top-k analysis (Figure 2-2). We provide a query panel to enable easy selection of ratings of interest (Figure 2-3). We also provide similar dashboards for each dimension and metric.

Our prototype is deployed to continually monitor the quality of these foundational platforms for about one year. Figure 3-

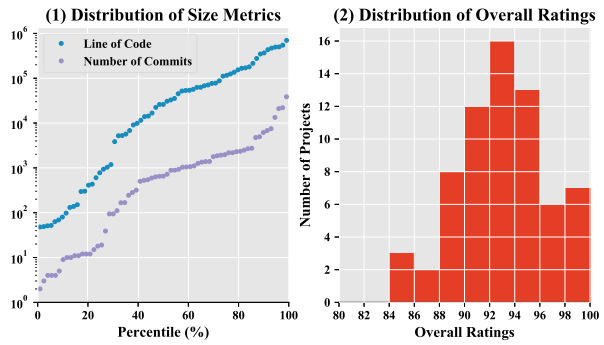

Fig. 3: Overview of Alibaba Projects Under Assessment

1 depicts the distribution of some size metrics (the number of lines of code and the number of commits) for the projects under assessment by quantile plots. We can observe that there are about 20% projects with more than 100,000 lines of code, but account for more than 80% code for all the projects. The number of commits for these projects also follows the same distribution. Figure 3-2 shows the histogram of the overall ratings of the projects. We can observe that the distribution follows a normal distribution (Shapiro-Wilk test p-values are larger than 0.05), which conforms to our intuition that the vast majority of the projects are of medium quality.

During the adoption by the Business Platform Unit, the quality assessment results are qualitatively validated from two aspects. First, we validate whether the overall quality ratings can be used to detect quality differences between different projects. We invite about 20 technical leaders in the Business Platform Unit to review the overall ratings for the software systems that they are responsible for. Specifically, they are asked to check the quality level of the software systems; the quality level is derived from the assessment results. Their feedback indicates that the quality level is consistent with their subjective opinions. Second, we validate whether the assessment results can be used to detect problematic code. Figure 4 shows the distribution of ratings for dimensions of the projects. We can observe that (1) there are some projects whose ratings of code duplication are much lower than the average, and (2) whose ratings of complexity and OO design have much room for improvement, especially for OO design. Therefore, we select some projects with the lowest ratings in these three dimensions and locate problematic code based on our assessment results. We observe some typical code smells in these projects, such as Type I Clone, Long Method, and God Class. We then contact the developers of these projects, and most of the reported problems are confirmed by these developers.

B. Adoption by Alibaba's R&D Efficiency Unit for group-wide software development

Alibaba's R&D Efficiency Unit is responsible for developing supporting platforms to improve the efficiency and

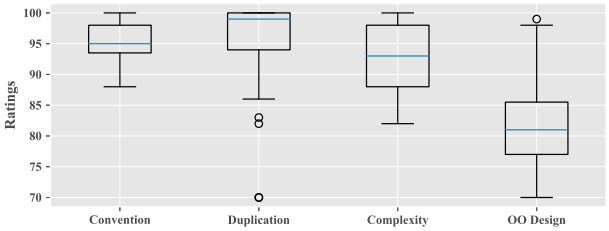

Fig. 4: Distribution of Ratings for Each Dimension

quality of software development in Alibaba. Our solution has been integrated with Aone, which is a one-stop collaborative platform for software development, from project planning and source code management to continuous integration and continuous deployment. More specifically, Aone applies the assessment capability of our solution in two scenarios: code search and code review.

The R&D Efficiency Unit develops and deploys a code search platform to promote code sharing and reuse in Alibaba, similar to code search in GitHub. Given a search query, the code search platform retrieves relevant code snippets from all the source code of repositories hosted by Aone. Traditionally, the candidate code snippets are ranked according to relevance to the given query. However, the quality of code snippets is ignored in this way, which brings the risk to spread low-quality code snippets across the code base. Therefore, the R&D Efficiency Unit integrates and deploys our solution into the code search platform so that it can take the quality of code snippets as a factor when ranking the candidate code snippets.

As a critical part of the code review practice in Alibaba, the R&D Efficiency Unit develops a code inspection platform based on our solution. This platform is similar to SonarQube [23], which can perform fully automatic reviews for quality assessment and provide powerful visualization for deep analysis. In fact, SonarQube also has been integrated into Aone, but development teams rarely adopt it in Alibaba due to its high false-positive rate under default settings. Moreover, it is not trivial to customize SonarQube to reduce false positives for two main reasons: (1) there are many coding rules that need to be customized; (2) there are no guidelines for customization, especially for those threshold-based rules.

V. Lessons Learned

In this section, we discuss the lessons that we learn during the development and deployment of our proposed industrial solution. We classify these lessons into technical aspects and social aspects, respectively.

A. Technical Aspects

1) Being Aware of Conflicts Among Requirements of Quality Assessment: During the stage of requirements analysis, we conduct various actions of gathering requirements, and collect a number of requirements. However, it is not unusual that some requirements are in conflict with each other. We need to

prioritize these requirements and make tradeoffs between the conflicting requirements.

For example, the accuracy and explainability of quality assessment are in conflict with each other. Many techniques, such as various complex regression and prediction models, can be applied to improve the accuracy of the assessment results, but the explainability of the assessment results is likely to be decreased in this manner. In particular, for utility-function-based rating, one of the reasons for us to select the linear function is that we can induce the original values from ratings by the inverse function, so that the assessment results can be interpreted directly according to the definition of metrics.

2) Paying Attention to Different Implementations of Software Metrics: During the implementation of collecting software metrics, we explore various existing tools, but find that they sometimes implement the same metric in different ways, and there are two main causes of such differences. First, the definition of a metric is sometimes quite flexible. For example, the definition of WMC does not specify how to set the weight, so people use different types of weight, such as unified weight, CC, and LOCm. Second, the definition of a metric is generally independent of the programming language used in the project under assessment. The divergence occurs when the metric is implemented for a specific language. For example, implementing most OO metrics encounters challenges in dealing with inner classes. Some people ignore the differences between inner and outer classes and deal with them in the same way. Others take inner classes as part of outer classes and eliminate inner classes by inlining.

However, different implementations are likely to have a critical impact on the measurement results and further work based on these results. Lincke et al. [24] pointed out that existing software metric tools implement OO software metrics differently, impacting the results of quality assessment based on these metrics. To address this issue, we first inspect the definitions of metrics and determine the implementation strategies for common attributes to avoid inconsistency. We then examine all the procedures influenced by the implementations for software metrics. For example, when we collect existing thresholds for metrics, we carefully inspect the implementation details of these metrics to check whether their implementations are consistent with ours.

B. Social Aspects

1) Increasing the Awareness of Assessment Tools: As Campbell et al. [25] argue, software development tools often suffer from the "deep discoverability" problem, which prevents developers from being aware of these tools. To increase awareness and adoption of assessment tools, we exploit three different approaches:

- **Posting on Online Developer Forums.** There is a popular technical forum in Alibaba for providing a collaborative environment to post development topics and questions for open discussion with other developers. We post some articles to introduce our tools and end up with thousands of page views.
- **Organizing Offline Technical Seminars.** We also organize some technical seminars to promote our tools. To attract more developers to attend, we invite some well-known experts to give talks in these seminars.
- **Supporting Programming Events.** To promote the adoption of the code inspection platform, we sponsor some company-wide hackathons in Alibaba. In these hackathons, the code inspection platform is employed as one source of rating for the code written by each participant.

2) Promoting the Adoption of Assessment Results: It is not trivial to convince developers to adopt the assessment results for improving software quality. As Jonathan et al. [26] pointed out, quality improvement often fails in practice due to the mishandling of the organizational and psychological factors. To address this issue, we explore two main strategies:

- **Being driven.** The quality improvement is initiated in a top-down manner, as developers are driven by managers to improve software quality. This strategy is adopted by the Business Platform Unit. In fact, the manager already has an opinion on the quality condition of the software systems. The main reasons why the manager introduces our quality assessment solution are attaining evidence from an independent assessor and having an external party to announce unpleasant truths. Therefore, the manager is willing to ask relevant developers to improve the quality of software according to the assessment results.
- **Being motivated.** Developers are self-motivated to improve software quality. This strategy is adopted by the R&D Efficiency Unit. More specifically, we aim to help developers gain reputation from high-quality code written by them, but avoid damaging their reputation due to low-quality code written by them. For example, the code search platform displays the owners of each candidate code snippet. The public exposure can increase the reputation of developers, and cause them to be more willing to write high-quality code to improve their rank in relevant queries.

VI. Conclusion

In this paper, we have presented our industrial solution for Alibaba's software quality assessment to ensure high sustainability of business growth and build engineering culture in Alibaba. We have summarized our solution's techniques and industrial adoption, along with the lessons learned during the development and deployment of our proposed industrial solution.

In future work, we plan to improve the efficiency and effectiveness of our solution. To address efficiency issues, we plan to implement the incremental measurement mechanism, inspired by the fact that most of the code usually remains unchanged in revisions and the measurement result of the unchanged code can be reused. To enhance the effectiveness, we plan to introduce some advanced techniques in our solution, such as using the GQM+Strategies [27] methodology to

align business strategies with software measurement and using architectural metrics to reflect the quality of high-level design.

VII. Acknowledgment

This work is supported by the National Key Research and Development Program of China (No.2017YFB1400603), National Natural Science Foundation of China under Grant (No.61772461, No.61825205, No.61772459, and No. 61529201), National Science and Technology Major Project of China (No.50-D36B02-9002-16/19), Natural Science Foundation of Zhejiang Province (No. LR18F020003 and No.LY17F020014) and the Alibaba-ZJU Joint Research Institute of Frontier Technologies, NSF under grants no. CNS-1564274 and CCF-1816615, and a grant from the ZJUI Research Program.

References

[1] C. Jones and O. Bonsignour, The economics of software quality. Addison-Wesley Professional, 2011.
[2] N. Leveson and C. Turner, "An investigation of the therac-25 accidents," Computer, vol. 26, no. 7, pp. 18–41, 1993.
[3] B. Boehm, J. Brown, and H. Kaspar, Characteristics of software quality. North-Holland, 1978.
[4] J. McCall, "Factors in software quality," US Rome Air development center reports, 1977.
[5] B. Behkamal, M. Kahani, and M. K. Akbari, "Customizing ISO 9126 quality model for evaluation of B2B applications," Information and Software Technology, vol. 51, no. 3, pp. 599–609, 2009.
[6] 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 Proceedings of the 2009 IEEE International Conference on Software Maintenance, 2009, pp. 531–534.
[7] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidl, A. Goeb, and J. Streit, "The Quamoco product quality modelling and assessment approach," in Proceedings of the 2012 IEEE/ACM International Conference on Software Engineering, 2012, pp. 1133–1142.
[8] M. Schnappinger, H. Osman, A. Pretschner, M. Pizka, and A. Fietzke, "Software quality assessment in practice: A hypothesis-driven framework," in Proceedings of the 2018 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 40:1–40:6.
[9] R. Van Solingen and E. Berghout, The Goal/Question/Metric method: a practical guide for quality improvement of software development. McGraw-Hill, 1999.
[10] M. Morisio, I. Stamelos, and A. Tsoukias, "Software product and process assessment through profile-based evaluation," International Journal of Software Engineering and Knowledge Engineering, vol. 13, no. 05, pp. 495–512, 2003.
[11] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, "A probabilistic software quality model," in Proceedings of the 2011 IEEE International Conference on Software Maintenance, 2011, pp. 243–252.
[12] F. Zhang, A. E. Hassan, S. McIntosh, and Y. Zou, "The use of summation to aggregate software metrics hinders the performance of defect prediction models," IEEE Transactions on Software Engineering, vol. 43, no. 5, pp. 476–491, 2017.
[13] 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.
[14] Alibaba, "Alibaba Java Coding Guidelines," accessed: 2019-04-10. [Online]. Available: https://alibaba.github.io/Alibaba-Java-Coding-Guidelines/
[15] A. Meneely, B. Smith, and L. Williams, "Validating software metrics: a spectrum of philosophies," ACM Transactions on Software Engineering and Methodology, vol. 21, no. 4, pp. 24:1–24:28, 2013.
[16] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," Information and Software Technology, vol. 55, no. 8, pp. 1397–1418, 2013.
[17] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in Proceedings of the 2009 IEEE/ACM International Symposium on Empirical Software Engineering and Measurement, 2009, pp. 367–377.
[18] K. Lochmann, "A benchmarking-inspired approach to determine threshold values for metrics," ACM SIGSOFT Software Engineering Notes, vol. 37, no. 6, pp. 1–8, 2012.
[19] "p3c-pmd," accessed: 2019-04-10. [Online]. Available: https://github.com/alibaba/p3c/tree/master/p3c-pmd
[20] H. Sajnani, V. Saini, J. Svajlenko, C. Roy, and C. Lopes, "SourcererCC: scaling code clone detection to big-code," in Proceedings of the 2016 IEEE/ACM International Conference on Software Engineering, 2016, pp. 1157–1168.
[21] "MaxCompute: petabyte-scale data warehousing," accessed: 2019-04-10. [Online]. Available: https://www.alibabacloud.com/product/maxcompute
[22] "Quick BI: business intelligence services on the cloud," accessed: 2019-04-10. [Online]. Available: https://www.alibabacloud.com/product/quickbi
[23] "Sonarqube," accessed: 2019-04-10. [Online]. Available: https://www.sonarqube.org
[24] R. Lincke, J. Lundberg, and W. Löwe, "Comparing software metrics tools," in Proceedings of the 2008 ACM International Symposium on Software Testing and Analysis, 2008, pp. 131–142.
[25] D. Campbell and M. Miller, "Designing refactoring tools for developers," in Proceedings of the 2008 Workshop on Refactoring Tools, 2008, pp. 9:1–9:2.
[26] J. Streit and M. Pizka, "Why software quality improvement fails:(and how to succeed nevertheless)," in Proceedings of the 2011 IEEE/ACM International Conference on Software Engineering, 2011, pp. 726–735.
[27] V. Basili, J. Heidrich, M. Lindvall, J. Munch, M. Regardie, and A. Trendowicz, "GQMˆ+ strategies – aligning business strategies with software measurement," in Proceedings of the 2007 International Symposium on Empirical Software Engineering and Measurement, 2007, pp. 488–490.