Code Quality in the Eyes of Crowd and in Terms of Metrics: A Comparative Study using Stack Overflow

Mohammad Masudur Rahman Chanchal K. Roy [1]Iman Keivanloo
University of Saskatchewan, [1]Queen's University, Canada
{masud.rahman, chanchal.roy}@usask.ca, [1]iman.keivanloo@queensu.ca

Abstract—**In StackOverflow, code examples are generally analyzed and subjectively evaluated by a large crowd of technical users. Given the growing interest of those examples to the community and their undeniable role in answers, we are motivated to study whether the subjective quality of those examples as perceived by StackOverflow actually matches with their metric-based quality. This is an important piece of information for the developers willing to reuse such examples. In this paper, we propose and evaluate a metric-based quality model for StackOverflow code examples by conducting an exploratory study where we analyze 160 code examples from 80 programming problems. Our quality model agrees with StackOverflow for 85.18% of the test examples in relative quality prediction, which is interesting, and the finding reveals effectiveness and reliability of the subjective evaluation by StackOverflow.**

Index Terms—**Objective code quality, readability, maintainability, subjective evaluation**

I. Introduction

Studies suggest that software developers spend about 19% of their development time in searching for relevant code on the web [4]. Stack Overflow, a popular online programming Q & A site, contains 10 millions questions on various programming topics, and they are used by a large community of 4 million technical users as of April 2014 [3]. In this site, users promote a question or an answer through up-voting when they find it useful or informative, and down-voting a post when they find its content erroneous or off-topic. The difference between up-votes and down-votes is considered as the *score* for the post.

During development and maintenance of a software product, software developers deal with different programming problems or challenges, and they frequently look into the programming issues posted on StackOverflow. The answers posted on the site often contain working code examples (i.e., code fragments) that solve particular programming problems or accomplish certain programming tasks. The developers find such examples helpful and reusable, and they also frequently apply them in their daily problem solving or learning activities. The posted code examples are generally viewed and subjectively evaluated (i.e., voting) by a large crowd of technical users. However, the objective quality of those examples is not often taken into consideration by the developers during reuse. Reusing or consulting with such code examples can be a threat if the promoted (i.e., up-voted) examples by the crowd maintain low quality in terms of objective code metrics.

Nasehi et al. [9] study the characteristics of the accepted answers of 163 programming questions from StackOverflow, and argue that the accepted answers are most likely to contain efficient and concise code examples accompanied by comprehensive textual description. Their study also reveals that the discouraged (i.e., extensively down-voted) answers from StackOverflow either do not contain code examples or miss the explanation about the code. Treude et al. [12] study which type of questions are answered correctly for most of the time, and suggest that the answers of the *code-review questions* (i.e., containing code examples) have the maximum acceptance rate of 92%. While these studies demonstrate the importance or the influence of StackOverflow code examples, the quality of those examples is not yet studied from metric-based point of view.

A number of existing studies focus on code level metrics for checking readability [5], reusability [11] or overall quality [7, 10] of the software code. Taibi [11] studies reusability of 33 open source projects based on code level metrics such as understandability, low complexity and modularity, and proposes a reusability model with different heuristic weights (i.e., importance) for different metrics. Mäntylä and Lassenius [8] conduct an empirical study to determine correlation between subjective evaluation and metric-based evaluation of software quality (w.r.t. code smells), and suggest that no evaluation alone is completely reliable. However, to date, no studies focus on the metric-based quality analysis of the StackOverflow code examples. In this research, we are interested to check whether the perceived quality of the code examples based on code level and associated metrics complies with StackOverflow votes for the corresponding examples. More specifically, we attempt to find out whether the promoted code example (e.g., Fig. 1-(a)) is actually preferable to the discouraged one (e.g., Fig. 1-(b)) for a programming problem in terms of different objective quality metrics. We formulate the following research question:

> **RQ**$_1$: Is the metric-based quality of a discouraged code example worse than that of a promoted code example?
> **RQ**$_2$: Can meta data about code examples separate promoted code examples from the discouraged ones?

We conduct an exploratory study using 160 highly promoted and highly discouraged code examples (i.e., code fragments) from 80 programming problems posted on StackOverflow. We apply five code level metrics– *readability*, *strength*, *concern*, *documentation* and *methodexist* and two associated metrics– *authorscore* and *editorscore*, and develop a objective quality

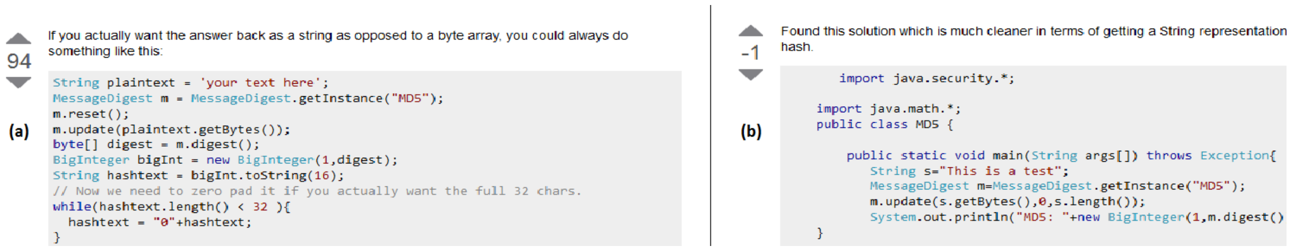

Fig. 1. (a) Promoted code example, (b) Discouraged code example

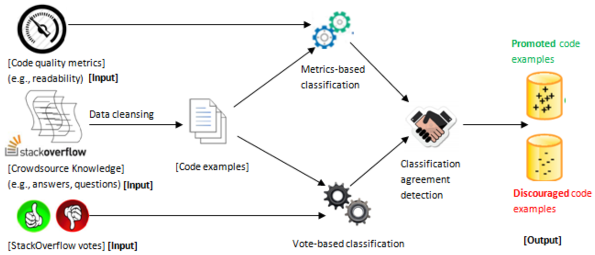

Fig. 2. Schematic diagram of the conducted study

model to perceive the quality of those code examples. Our model agrees with StackOverflow for 85.18% of the test examples in *relative quality prediction*, which is promising, and it reveals the effectiveness and reliability of the subjective evaluation by StackOverflow. We also investigate the examples for which the model does not agree with StackOverflow on quality analysis, and finally answer the research question.

II. Methodology

Fig. 2 shows the schematic diagram of our conducted study. In this section, we discuss the detailed design of the study, our proposed metrics and proposed model for objective quality analysis of StackOverflow code examples.

A. Data Collection

We use StackExchange Data API [1] (i.e., provides access to the data of several StackExchange sites), and collect 80 programming questions along with their answers from StackOverflow which are related to *android platform* and *android applications*. It should be noted that each of those questions has more than *ten answers* that contain *code examples*. The idea is to collect such questions which are both widely discussed and related to coding. We choose three top-ranked and three lowest-ranked answers based on StackOverflow votes for each of those questions. We then extract code examples from the raw HTML of those StackOverflow answers, and manually analyze them. We look for trivial examples such as *novice's examples* (i.e., intended for novice users), *abstract examples* (i.e., too generic) or *small examples* (i.e., contains less than three lines of code), and discard them in order to develop a list of qualified examples. Finally, as discussed in Section II-B below, we select two code examples out of the remaining examples for each question, and one of them is from highly promoted (i.e., up-voted) answer and the other is from highly discouraged (i.e., down-voted) answer.

B. Vote Based Classification

In StackOverflow, the technical merit or quality of a programming answer is recognized in terms of votes [9], and code examples are generally posted as a part of those answers. Existing studies [9, 12] report and explain the undeniable role of code examples in the promotion or demotion of the posted answers. Thus the votes cast by thousands of technical users of StackOverflow for those answers can also be considered to approximate the subjective quality of the code examples contained by those answers. Given that the subjective perception of quality may vary, we choose the code examples of two extreme quality perceptions– highly promoted and highly discouraged. We examine the *scores* (i.e., difference between up-votes and down-votes) achieved to date (i.e., November 01, 2014) by different answers for each of the 80 programming questions. We then choose the corresponding code examples from the highly promoted (i.e., up-voted) and the highly discouraged (i.e., down-voted) answers for each question, and classify them as *highly promoted* and *highly discouraged* code examples respectively.

C. Objective Quality Metrics

StackOverflow code examples are generally posted either as complete methods or code segments containing a few lines, and the entire *class-structure* is often unlikely. Therefore, most of the available code quality metrics such as *object-oriented complexity* metrics are not applicable for those examples. In this section, we discuss five code related metrics and two associated metrics used for the study.

Readability (R): Readability of software code refers to a human judgement of how easy the code is to understand [5]. Reading (i.e., understanding) code is one of the most time-consuming components of all software maintenance activities, and thus, readability is directly related to software maintainability. The baseline idea is– the more readable or understandable the code is, the easier it is to reuse and maintain in the long run. Buse and Weimer [5] propose a code readability model trained on human perception of readability or understandability. The model uses different textual source features (e.g., length of identifiers, number of comments, line length) that are likely to affect the humans' perception of readability, and predicts a *readability score* on the scale from zero to one, inclusive, with one describing that the source code is highly readable. We use the readily available library[2] by Buse and Weimer [5] for calculating the *readability metric* of the code examples.

Author Rank (AR) & Editor Rank (ER): As existing studies [6] suggest, we believe that the expertise of the author or an editor of a code example is likely to influence its quality. StackOverflow provides different incentives to the users who actively contribute to the body of knowledge by asking important questions, posting helpful answers or adding informative comments. One of those incentives is *Reputation* (i.e., an estimation of overall contribution to the site) which can be often considered as an approximation of one's expertise. In order to determine *author rank* and *editor rank* of a code example, we collect the *reputation scores* of the author and the last editor of that example. We then normalize these scores against the *maximum user reputation* from StackOverflow, and provide both metrics on the scale from zero to one, where zero denotes the least expertise and vice versa.

Strength (S) and Concern (W): Mäntylä and Lassenius [8] conduct an empirical study on software evolvability by employing subjective and metric-based identification of code smells. They analyze agreement level between the two evaluations, and argue that neither technique alone is enough to detect all smells. Similarly, we can conjecture that code level metrics are not sufficient enough to discover all types of defects, inefficiencies or possible scopes for improvement in the code examples. StackOverflow facilitates to include the subjective evaluations for each code example in the form of comments which often contain invaluable and insightful analysis about its code level quality. While one can argue that the comments are merely based on subjective viewpoints, we note that they also contain objective observations which can be considered to derive metrics describing the soundness of the code example. We leverage the objective observations to identify the strengths and weaknesses of the code example. Basically, we analyze all the comments about a code example against the code and count their numbers discussing about positive aspects (i.e., strength) and negative aspects (i.e., weakness) of the code. Then we normalize the *strength* and *weakness* measures using *maximum comment count* among the code examples of the *same question* as follows.

$$S_i = \frac{S_{i,count}}{max(TC_i)}, \quad W_i = \frac{W_{i,count}}{max(TC_i)} \tag{1}$$

Here, $S_{i,count}, W_{i,count}$ and TC_i denote the positive comments count, negative comments count and total comments count of a code example respectively. Both the *strength* and *weakness* metrics provide a normalized score on the scale from zero to one, where zero represents the least measure of each metric.

Rule Violation (RV): Traditional metric-based quality evaluation is dominated by code analysis tools, and they try to analyze the code against a certain set of *recommended rules*. Most of the tools concentrate on particular aspects of code. For example, *CheckStyle* focuses on conventions, *PMD* on bad practices and *FindBugs* focuses on potential bugs or threats. Thus, rules and standards of one tool may vary from another, and the tools are no way competitive rather complementary. Given the facts about the tools, using any single one may not serve our purpose of detecting rule violations, and thus we use *sonarQube*[1] which combines the rules and standards of *PMD, FindBugs, CheckStyles* and so on. We collect three types of violations – critical, major and minor, in the code examples and determine *violations per source line* for each of them. Lochmann and Heinemann [7] propose a rule-based quality model for the comprehensive quality assessment of a complete software project using the rules extracted from static code analysis tools. However, given the coding structure and size of StackOverflow code examples, we hypothesize that *violation per source line* is an important and credible metric to estimate the relative quality of the code examples. In order to preserve uniformity with other metrics, we normalize *violation per source line* for each code example.

D. Metric-Based Quality Model

We consider readability, author's expertise, adherence to the best coding practices, identified issues and threats in the code examples to estimate the quality of the examples with the focus on their reusability. We randomly select 50 code examples from 25 programming questions in the dataset, analyze their quality, and manually label them either as *promoted* or *discouraged*. Then we use those labeled examples along with their computed metrics (i.e., Section II-C), and use logistic regression-based classifier from *Weka* to determine the relative predictive power of the proposed metrics. It should be noted that we use Odd Ratio [6] of each metric, which is a logarithmic transformation of the metric coefficient in the regression equation of the classifier, and tune them under controlled iterations to determine the predictive power (i.e., importance) of the metric. Since, we are interested in determining the relative quality (i.e., without a threshold) of two code examples of the same question, we ignore the intercept of the equation, and develop the following quality model.

$$\begin{aligned} Q_i = 3.0043 \times R_i + 4.3067 \times AR_i + 8.04884 \times S_i \\ +0.5078 \times W_i + 0.5812 \times RV_i \end{aligned} \tag{2}$$

In the model, we find *readability*, *author rank*, and *strength* as the most dominating features (i.e., metrics), whereas *weakness* and *rule violation* as the least influencing ones. It should be noted that the first three of the metrics are positive factors for software code quality (i.e., improves quality) and the rest two are negative factors (i.e., degrades quality). Since, we are interested in the relative quality assessment of the code examples, we use the logarithmic transformation that provides different Odd Ratios (i.e., weights) of the metrics within the same range (i.e., greater than zero) and makes the quality estimates more comprehensive.

III. Results and Discussions

In our experiment, we use the proposed quality model to estimate the quality of 110 code examples against 55 programming questions (dataset can be found online[2]). It should be noted that we use the quality estimates to perform

[1] http://www.sonarqube.org/

[2] www.usask.ca/∼masud.rahman/ss/expdata

TABLE I
EXPERIMENTAL RESULTS

Metrics	APC[1]	A[2]	D[3]
{R, AR}	30(55)	54.55%	45.45%
{R, S}	39(55)	70.91%	29.09%
{R, W}	29(55)	52.72%	47.28%
{R, AR, S}	**42(55)**	**76.36%**	**23.64%**
{R, S, W}	41(55)	74.54%	25.45%
{R, AR, S, W}	**43(55)**	**78.18%**	**21.82%**
{R, S, W, RV}	41(55)	74.54%	25.45%
{R, AR, S, W, RV}	**43(55)**	**78.18%**	**21.82%**

[1]No. of example pairs for which relative quality evaluation matches with that of StackOverflow,
[2] % of agreement, [3] % of disagreement

TABLE II
METRICS CORRELATION

Metrics	CR[1]	P[2]	Metrics	CR	P
R, AR	-0.1767	0.0647	AR, S	0.1164	0.2258
R, S	-0.0916	0.3412	AR, W	0.1045	0.2772
R, W	-0.0512	0.5954	S, W	-0.2321	0.0147

[1]Correlation, [2]p-value

the comparative analysis among the two code examples of the same question. The idea is to determine whether a code example promoted by StackOverflow is actually of better code quality than the one which is discouraged by it from metric-based point of view. Table I shows the results of our preliminary experiments using Equation (2), where the metric-based relative quality of the code examples agrees with that of StackOverflow at best for 43 (78.18%) out of 55 questions. It also shows how different component quality metrics can influence the estimated overall quality of the code examples, and the empirical findings show that *readability*(R), *author rank*(AR) and *strength*(S) are the most effective metrics for relative quality analysis when they are considered in combination. We note that *weakness* and *rule violation* metrics have a little or no influence to the proposed model, which refutes our initial assumption.

RQ: Is the code level quality of a discouraged code example worse than that of a promoted code example? Our preliminary results (Table I) indicate that the code quality of discouraged code examples is worse than that of promoted code examples in 78% of the examples. Based on the subjective evaluation (e.g., score per day) by StackOverflow, we estimated the relative quality of the two code examples against each of our selected questions, and determined the *promoted* and the *discouraged* ones. In order to determine their code level quality, we collected the target quality metrics (Section II-C) of each code example, and applied them to the proposed quality model. The model provides an estimate about the quality of each example, and we used those estimates to determine the relative code quality of the two code examples against a question. Then we compared the metric-based relative quality against the corresponding relative quality obtained from subjective evaluation. The result shows that the discouraged code examples are of inferior quality in terms of code metrics to the promoted ones for 43 (78.18%) test cases out of 55 cases.

According to our experiment, the two classifications agree mostly, about 78%; however, the complete agreement may not be possible. We investigated the 12 cases (24 code examples and their metrics) for which our quality model does not match with StackOverflow, and found a few issues or scenarios. First, most of them do not contain comments given that metrics (e.g., strength and weakness) from the comments play major parts in our model, and the model does not perform well for those cases (i.e., 9 cases). Second, our model does not use any threshold to describe a code example either as promoted or discouraged, rather it uses relative quality analysis which may not be effective all the time. For example, if there is a little difference in the quality estimate of two code examples, the model still determines the promoted and discouraged ones; however, both of them should be considered either as promoted or discouraged in practical. We found one such case in the experiment. Third, StackOverflow contains some code examples, which are highly simplified with little technical merit, and are often intended for preliminary learning, and they are also highly voted. Our model does not perform well in that case (i.e., 3 cases).

Given that *code quality* of the software code is a multi-faceted term [10], we focus on the quality analysis to determine the *reusability* of a code example. *Readability*, *strength*, *weakness* and *rule violation* metrics are greatly related to comprehensibility, efficiency, security, maintainability and other attributes (i.e., quality) of the software code that stimulate its reuse [5]. In our experiment, we found *strength*(e.g., weight 8.05) and *readability* (e.g., weight 3.00) are the most predictive metrics while combined for quality analysis. On the other hand, *weakness* and *rule violation* metrics are found not predictive. We can speculate that *rule violation* metric may not be properly applicable for StackOverflow code examples due to their fragmented nature, and *weakness* metric may need to be refined for effective use; however, we need to experiment with more data to reach a conclusion.

IV. CONCLUSION & FUTURE WORKS

Given the growing interest of StackOverflow community to the code examples, we attempt to determine whether their subjective evaluation by the community agrees with the metric-based evaluation. We conduct an exploratory study with 110 representative code examples against 55 programming questions, and found that the subjective evaluation agrees with the metric-based evaluation for 78% of code examples. The finding is quite promising, and it reveals the effectiveness of StackOverflow votes. It also has the potential to encourage more research in the quality analysis of the code examples often found in the programming Q & A sites, and our developed quality model can assist the developers in reusing code examples with informed knowledge of metric-based quality.

REFERENCES

[1] StackExchage API. URL http://data.stackexchange.com/stackoverflow.
[2] Readability Library. URL http://www.arrestedcomputing.com/readability.
[3] StackOverflow. URL http://en.wikipedia.org/wiki/Stack_Overflow.
[4] J. Brandt, P.J. Guo, J. Lewenstein, M. Dontcheva, and S.R. Klemmer. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proc. SIGCHI*, pages 1589–1598, 2009.
[5] R.P.L. Buse and W.R. Weimer. Learning a Metric for Code Readability. *TSE*, 36 (4):546–558, 2010.
[6] C. Le Goues and W. Weimer. Measuring Code Quality to Improve Specification Mining. *TSE*, 38(1):175–190, 2012.

[7] K. Lochmann and L. Heinemann. Integrating Quality Models and Static Analysis for Comprehensive Quality Assessment. In *Proc. of WETSoM*, pages 5–11, 2011.

[8] M.V. Mäntylä and C. Lassenius. Subjective Evaluation of Software Evolvability using Code Smells: An Empirical Study. *ESE*, 11(3):395–431, 2006.

[9] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What Makes a Good Code Example -A Study of Programming Q and A in Stackoverflow. In *Proc. ICSM*, pages 25 –35, 2012.

[10] M.S. Rawat, A. Mittal, and S.K. Dubey. Survey on Impact of Software Metrics on Software Quality. *IJACSA*, 3(1), 2012.

[11] F. Taibi. Reusability of Open-Source Program Code: A Conceptual Model and Empirical Investigation. *SE Notes*, 38(4), 2013.

[12] C. Treude, O. Barzilay, and M.A. Storey. How Do Programmers Ask and Answer Questions on the Web? (NIER Track). In *ICSE*, pages 804–807, 2011.