StackOverflow Code Quality Model: An Exploratory Study

Mohammad Masudur Rahman Chanchal K. Roy [†]Iman Keivanloo
University of Saskatchewan, [†]Queen's University, Canada
{masud.rahman, chanchal.roy}@usask.ca, [†]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 the community to those examples 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. 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 carefully selected code examples from 80 programming problems. Our quality model agrees with StackOverflow for 87.03% of the test examples in relative quality prediction, which is promising, and the finding reveals effectiveness and reliability of the subjective evaluation by StackOverflow.**

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

I. Introduction

StackOverflow [4], a popular social programming Q & A site, has a large community of 2.7 million technical users, and it covers about 7.1 million programming questions from different programming domains (e.g., android, API) and languages (e.g., C, Java, C#). In StackOverflow, users promote a question or an answer through up-voting when they find it useful and 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 a post.

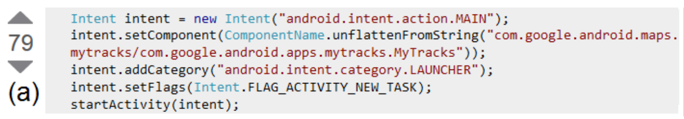
During development and maintenance of a software product, software developers deal with different programming problems or challenges, and they frequently look into the programming issues posted on StackOverflow. The answers posted on the site often contain working code examples. A code example is a code snippet that solves a particular programming problem or accomplishes a certain programming task [11]. The developers find such examples helpful and reusable, and they also frequently apply them in their daily problem solving or learning activities. Although the code examples are subjectively evaluated by StackOverflow, 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. Thus, we are interested to study if there exists a relation between the subjective evaluation of those examples and their objective evaluation based on metrics.

Nasehi et al. [11] 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. Treude et al. [14] 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.

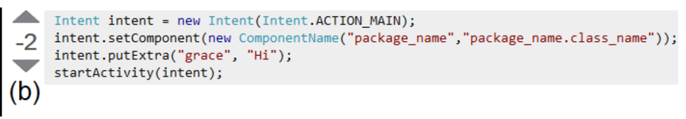
A number of existing studies focus on code level metrics for checking readability [6], reusability [13] or overall quality [9] of the software code. Taibi [13] studies reusability of 33 open source projects based on code level metrics such as understandability, low complexity and modularity, and proposes a reusability model with different heuristic weights (i.e., importance) for different metrics. Mäntylä and Lassenius [10] 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 objective code metrics. We formulate the following research question:

RQ: Is the metric-based quality of a discouraged code example worse than that of a promoted code example?

We conduct an exploratory study using 160 highly promoted and highly discouraged code examples from 80 programming problems posted on StackOverflow. We apply five code level metrics– *readability*, *strength*, *concern*, *documentation* and *methodexist* and two associated metrics– *author rank* and *editor rank*, and develop an objective quality model to perceive the quality of those code examples. Our model agrees with StackOverflow for 87.03% of the test examples in *relative quality prediction*. This preliminary result is promising, and it shows that the subjective evaluation by StackOverflow is effective and reliable. We also investigate the examples for which the model does not agree with StackOverflow on quality analysis, and finally answer the research question.



```
Intent intent = new Intent("android.intent.action.MAIN");
intent.setComponent(ComponentName.unflattenFromString("com.google.android.maps.
mytracks/com.google.android.apps.mytracks.MyTracks"));
intent.addCategory("android.intent.category.LAUNCHER");
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
startActivity(intent);
```



```
Intent intent = new Intent(Intent.ACTION_MAIN);
intent.setComponent(new ComponentName("package_name","package_name.class_name"));
intent.putExtra("grace", "Hi");
startActivity(intent);
```

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

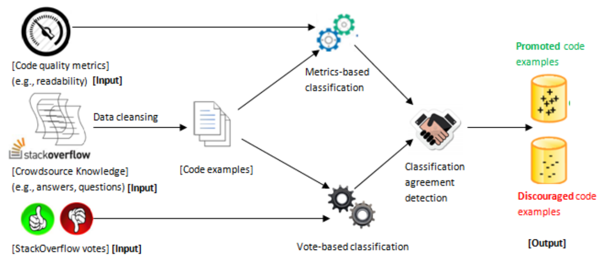

[Code quality metrics]
(e.g., readability) [Input]
Data cleansing
stackoverflow
[Crowdsource Knowledge]
(e.g., answers, questions) [Input]
[Code examples]
Metrics-based classification
Classification agreement detection
Promoted code examples
Discouraged code examples
[Output]
[StackOverflow votes] [Input]
Vote-based classification

Fig. 2. Schematic diagram of the conducted study

II. Methodology

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

A. Data Collection

In order to collect data [2], we use StackExchange Data API [1] that provides access to the data of several StackExchange sites. For this study, we wanted to collect the code examples which are widely viewed or evaluated on StackOverflow, and the logical first step was to identify such programming questions. We discover 80 such questions in the domain of *android applications* from StackOverflow using the API, where each of the questions has more than *ten answers* that contain *code examples*. 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., extensively 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 [11], and code examples are generally posted as a part of those answers. Existing studies [11, 14] report and explain the undeniable role of code examples in the promotion or demotion of the posted answers. Thus the votes cast 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 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 as a code snippet containing a few lines. Therefore, some of the code quality metrics such as *object-oriented complexity* metrics are not applicable. In this research, we use a list of five code level metrics and two associated metrics that are proposed by earlier studies for code quality analysis.

Readability (R): Readability of software code refers to a human judgement of how easy the code is to understand [6]. 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 [6] 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 [3] by Buse and Weimer [6] for calculating the *readability metric* of the code examples.

Author Rank (AR) & Editor Rank (ER): Existing studies [8] suggest 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 [5]. 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 (C): Mäntylä and Lassenius [10] conduct an empirical study on software evolvability by employing subjective and metric-based identification of code smells. They analyze agreement level between the two techniques, and argue that neither technique alone is enough for detecting all the smells. Similarly, we can conjecture that code level metrics are not sufficient enough to discover all defects or inefficiencies and possible scopes for improvement in the code examples. In StackOverflow, users post their subjective observations on the code examples in the form of feedback comments which often contain invaluable and insightful analysis about the code level quality of those examples. While one can argue that the comments are merely based on subjective viewpoints, we interestingly note that they also contain objective observations which can be exploited to approximate the utility (i.e., *strength*) and more specifically the weakness (i.e., *concern*) of a code example. In order to collect meaningful comments, we choose only those comments for a code example which have obtained at least *one* up-vote to date from the technical crowd. The baseline idea is – the up-voted comments are more likely to contain objective observations as they are recognized by other users. We then extract each of the individual sentences from those comments, and apply a state-of-the-art sentiment analyzer [12] on them. The tool analyzes each of the words in the sentence, determines their positivity or negativity based on a trained model, and finally returns the overall sentiment for the sentence on this scale– *very positive* (i.e., "4"), *positive* (i.e., "3"), *neutral* (i.e., "2"), *negative* (i.e., "1"), and *very negative* (i.e., "0"). We adapt the scale so that *neutral* point equals to "0", and then estimate *strength (S)* and *concern (C)* of a code example as follows:

$$P = \sum_{i}^{M} PS_i,\ N = \sum_{j}^{T} NS_j,\ S = \frac{P}{P+|N|},\ C = \frac{N}{P+|N|}$$

Here PS, NS, P and N denote positive sentence set, negative sentence set, total positivity score and total negativity score respectively for a code example. While *strength (S)* refers to an estimation of *positivity* about the example, *concern (C)* reports a heuristic quantification of the *defects, concerns* or *limitations* of the example that are identified by the technical crowd. Both metrics are normalized for the sake of simplicity in the development of the quality model.

Documentation Existed (DE): According to Nasehi et al. [11], discouraged answers in StackOverflow often miss the explanation about the contained code, which suggests that documentation is a complementary part of a code example. Wong et al. [15] suggest that the documentation for a code example can be found in a few sentences that precede the example in the answer. We manually analyze the answer posts, and determine *documentation existed (DE)*, a binary metric, for each of the code examples in the dataset.

IsAMethod (IM): In StackOverflow, each code example is posted either as an entire method or a code segment. We believe that the code examples with entire methods are easy to integrate into the development code for similar programming tasks, and thus they require less effort for further maintenance. We thus propose a binary ad-hoc metric, called, *isamethod* to capture this maintainability aspect of the code examples. We manually analyze each of the examples, look for method definitions, and determine the metric for the examples.

TABLE I
COEFFICIENTS IN LOGISTIC REGRESSION

Metric	Weight	Metric	Weight
strength	140.602	*author rank*	12.412
editor rank	1.945	*readability*	-0.071
concern	-2.899	*isamethod*	0.128
documentation existed	1.253		

D. Metric-Based Quality Model

We collect five code level and related metrics– *readability, strength, concern, documentation existed* and *isamethod* and two associated metrics– *author rank* and *editor rank* of StackOverflow code examples, and apply them in the development of an *objective quality model* for those examples. Out of 160 code examples, we use 106 examples (i.e., 66%, through dataset splitting) for training where we employ logistic regression on the training set using *Weka*, a machine learning workbench. Due to the exploratory nature of this study, we perform a step-wise logistic regression where different metrics are gradually added to the model in order to optimize its overall fit [7]. For example, when we add only *author rank*, *editor rank* and *strength* to the model, we experience 75.93% classification accuracy with the test set. However, when we also add *concern* and *documentation existed* to the model, we find the model with the maximum predictive power (i.e., 87.03% accuracy). Deletion or addition of any other metric to the model does not result into further improvement of the model. From Table I, we also note that the remaining two metrics–*readability* and *isamethod* have the least weights (i.e., collected from a training step with all seven metrics). We thus finalize our model for objective quality analysis of StackOverflow code examples by employing these five metrics– *strength, author rank, editor rank, documentation existed* and *concern* as follows:

$$Class(example\ e) = \frac{1}{1+e^{-Q(e)}},\ with$$

$$Q(e) = -2.270 + 12.746 \times AR(e) + 1.895 \times ER(e)$$

$$+140.113 \times S(e) - 2.899 \times C(e) + 1.240 \times DE(e)$$

In the model, as the feature weights suggest, we find *strength* and *author rank* as the most predictive features (i.e., metrics), whereas *editor rank* and *documentation existed* are found relatively less predictive. It also should be noted that *concern* has a negative effect on the model. Thus, the model can predict a code example as *highly promoted* only if StackOverflow community expresses a higher sense of positivity (i.e., *strength*) about the example, the author or the editor of the example is reputed (i.e., *author rank*, *editor rank*), and the example code is associated with comprehensive documentation.

III. EVALUATION RESULTS AND DISCUSSIONS

Classification Results: In order to evaluate our proposed model, we develop a *test set* containing 54 code examples (i.e., 34%, through dataset splitting) from the dataset. It should be noted that this set does not overlap with the *training set*. Table II reports how the performance of our model against the *test set* evolves with the step-wise addition of different metrics. For example, being the most dominating predictor,

TABLE II
EXPERIMENTAL RESULTS

Metric Combination	RMSE[1]	κ[2]	F_1[3]	AUC[4]	CA[5]
{S}	0.43	0.42	0.68	0.71	70.37%
{S, AR}	0.40	0.53	0.75	0.88	75.93%
{S, AR, C}	0.33	0.74	0.87	**0.91**	**87.03%**
{S, AR, C, ER}	0.33	0.74	0.87	0.90	87.03%
{S, AR, C, DE}	0.34	0.74	0.87	**0.92**	**87.03%**
{S, AR, C, ER, DE}	0.34	0.74	0.87	0.91	87.03%

[1]Root Mean Squared Error, [2]Kappa statistic, [3]F_1 score, [4]Area Under ROC Curve, [5]Classification Accuracy

TABLE III
METRIC CORRELATION

Metrics	CR[1]	P[2]	Metrics	CR	P
S, AR	0.13	0.18	S, C	-0.19	0.05
AR, C	-0.50	0.00	AR, DE	0.38	0.00
AR, ER	0.11	0.24	C, DE	-0.24	0.02

[1]Correlation coefficient, [2]p-values

strength alone can predict the quality of 70.37% of the samples correctly. However, the model performs poor in terms of *Root Mean Squared Error (RMSE), Area Under Curve (AUC)* or κ-statistic. We then add the next best (i.e., based on weights in Table I) predictors–*author rank* and *concern*, and the model improves significantly in terms of all performance metrics. For example, the model predicts correctly for 87.03% of the test samples with an improved κ-statistic (e.g., 0.74) and a reduced error rate (e.g., 0.33). We also add two other metrics–*editor rank* and *documentation existed* having moderate predictive power, which does not result into significant performance improvement for the model with our *test set*. In order to investigate this finding, we determine the correlation among the metrics (Table III). Although we did not experience any significant correlation between these two metrics with the previously added dominating metrics, we found an interesting scenario about *reputed author*. From Table III, we note that *author rank* is moderately correlated with *document existed* and *concern*, which indicates that the code example written by a reputed user in StackOverflow is likely to be accompanied by a comprehensive explanation, and the community would generally have a reduced sense of negativity (i.e., *concerns*) towards that example.

Goodness of Fit Test: We evaluate the goodness of fit of our model using two widely used statistical tests– *Chi-square statistic* [7] and *Classification table*. Using *Wald test*, we find that no individual metrics, not even *strength* has a significant overall effect on the model in isolation. On the other hand, when we consider *author rank, editor rank, strength* and *concern* in combination, we experience a significant effect (*chi-square=15.6, P=0.003 with df=4*). We also check the overall fit of our model compared to a null model (i.e., without any predictor variables) using *log likelihood test*, and found that the model fits significantly (*chi-square=73.22, P<.000 with df=5*) better than an empty or null model. We also analyze the test sample classification table, and find that the classifier provides a *sensitivity* of 0.92 and a *specificity* of 0.83, which indicates that the model fits significantly with the test data.

Answer to Research Question: Based on the subjective evaluation by StackOverflow, we estimated the relative quality (i.e., *promoted* and *discouraged*) of the two code examples for each of our selected questions, and our trained model correctly predicts such quality for 47 (i.e., 87.03%) out of 54 examples in the *test set*. The finding essentially answers our research question that overall objective quality of *promoted* code examples is better than that of *discouraged* examples at 87.03% of the times. We also investigate the examples for which the model fails to predict correctly. We observe that five out of seven such test examples miss the values for dominating metrics such as *strength* and *concern*, and the rest two contain either equal or counter-intuitive values for those metrics, which indicates a certain bias of the model. Thus, although the model performs significantly well with the *test set*, it needs further training with more code examples in order to reduce the suppressor effects and to develop a more balanced objective model. One can also argue about the objectivity of *strength* and *concern* metrics as they are derived from feedback comments. However, we selected those comments carefully, and then accumulate *positivity* and *negativity* about an example from the large crowd using a reliable sentiment analyzer [12] in order to reduce subjectivity from those metrics.

IV. CONCLUSION & FUTURE WORK

Given the growing interest of StackOverflow community to the code examples, in this research, we study whether their subjective quality as perceived by the community matches with an objective quality evaluation. We propose and evaluate a metric-based quality model for StackOverflow code examples by conducting an exploratory study with 160 code examples collected from 80 programming problems. The model agrees with StackOverflow in quality prediction for 87.03% of the test examples, which reveals the effectiveness and reliability of StackOverflow votes. While the model can assist the developers in reusing StackOverflow code examples with informed knowledge about their quality, our study also has the potential to encourage further research in the quality analysis of the code examples from other programming Q & A sites.

REFERENCES

[1] StackExchage API. URL http://data.stackexchange.com/stackoverflow.

[2] Experiment data. URL www.usask.ca/~masud.rahman/ss/expdata.

[3] Readability Library. URL http://www.arrestedcomputing.com/readability.

[4] StackOverflow. URL http://en.wikipedia.org/wiki/Stack_Overflow.

[5] A. Bosu, C.S. Corley, D. Heaton, D. Chatterji, J.C. Carver, and N.A. Kraft. Building Reputation in StackOverflow: An Empirical Investigation. In *Proc. MSR*, pages 89–92, 2013.

[6] R.P.L. Buse and W.R. Weimer. Learning a Metric for Code Readability. *TSE*, 36 (4):546–558, 2010.

[7] K. Kevic and T. Fritz. Automatic Search Term Identification for Change Tasks. In *Proc. ICSE*, pages 468–471, 2014.

[8] C. Le Goues and W. Weimer. Measuring Code Quality to Improve Specification Mining. *TSE*, 38(1):175–190, 2012.

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

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

[11] 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.

[12] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank. In *Proc. EMNLP*, 2013.

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

[14] 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.

[15] E. Wong, J. Yang, and L. Tan. AutoComment: Mining Question and Answer sites for Automatic Comment Generation. In *Proc. ASE*, pages 562–567, 2013.