Investigating the Quality Aspects of Crowd-Sourced Developer Forum: A Case Study of Stack Overflow

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Saikat Mondal

Permission to Use

Abstract

Technical question and answer (Q&A) websites have changed the way how developers seek information on the web and become more popular due to the shortcomings in official documentation and alternative knowledge sharing resources. Stack Overflow (SO) is one of the largest and most popular online Q&A websites for developers where they can share knowledge by answering questions and learn new skills by asking questions. Unfortunately, a large number of questions (up to 29%) are not answered at all, which might hurt the quality or purpose of this community-oriented knowledge base. In this thesis, we first attempt to detect the potentially unanswered questions during their submission using machine learning models. We compare unanswered and answered questions quantitatively and qualitatively. The quantitative analysis suggests that topics discussed in the question, the experience of the question submitter, and readability of question texts could often determine whether a question would be answered or not. Our qualitative study also reveals why the questions remain unanswered that could guide novice users to improve their questions. During analyzing the questions of SO, we see that many of them remain unanswered and unresolved because they contain such code segments that could potentially have programming issues (e.g., error, unexpected behavior), unfortunately, the issues could always not be reproduced by other users. This irreproducibility of issues might prevent questions of SO from getting answers or appropriate answers. In our second study, we thus conduct an exploratory study on the reproducibility of the issues discussed in questions and the correlation between issue reproducibility status (of questions) and corresponding answer meta-data such as the presence of an accepted answer. According to our analysis, a question with reproducible issues has at least three times higher chance of receiving an accepted answer than the question with irreproducible issues. However, users can improve the quality of questions and answers by editing. Unfortunately, such edits may be rejected (i.e., rollback) due to undesired modifications and ambiguities. We thus offer a comprehensive overview of reasons and ambiguities in the SO rollback edits. We identify 14 reasons for rollback edits and eight ambiguities that are often present in those edits. We also develop algorithms to detect ambiguities automatically. During the above studies, we find that about half of the questions that received working solutions have negative scores. About 18% of the accepted answers also do not score the maximum votes. Furthermore, many users are complaining against the downvotes that cast to their questions and answers. All these findings cast serious doubts on the reliability of the evaluation mechanism employed at SO. We thus concentrate on the assessment mechanism of SO to ensure a non-biased, reliable quality assessment mechanism of SO. In this study, we compare the subjective assessment of questions with their objective assessment using 2.5 million questions and ten text analysis metrics. We also develop machine learning models in order to classify the promoted and discouraged questions and to predict them during their submission time.

We believe that findings from our studies and proposed techniques have the potential to (1) help the users to ask better questions with appropriate code examples, and (2) improve editing and assessment mechanism of SO to promote better content quality.

Acknowledgements

At first, I like to praise Almighty God, the most gracious and most merciful, who gave me the ability to carrying out this work. Next, I would like to express my sincerest appreciation to my supervisor Dr. Chanchal K. Roy for his continuous support, guidance, motivation, and remarkable endurance during this thesis work. Without his support, this work would have been unthinkable.

I would like to thank Dr. Nurul Chowdhury, Dr. Nadeem Jamali, Dr. Banani Roy and Dr. Zadia Codabux for their willingness to take part in the advisement and evaluation of my thesis work. I would also like to thank them for their valuable time, suggestions and insights.

I would also wish to express my gratefulness to Dr. Mohammad Masudur Rahman, Dr. Gias Uddin for extended discussions, valuable suggestions, passionate participation, and input, which have contributed greatly to the improvement of the thesis.

I would like to express my special appreciation and thanks to the anonymous reviewers for their valuable comments and suggestions on improving the papers produced from this thesis.

I express my heartiest gratitude to my late father Tarak Chandra Mondal and my mother Tuku Mondal who are the architects of my life. Their endless sacrifice, unconditional love, and constant good wishes have made me reach this stage of my life. I would also like to thank my mother-in-law Kabita Biswas, father-in-law Bimal Chandra Lasker, and uncle Kishore Kumar Golder for their constant well wishes and inspirations in this thesis work. I am also thankful to my elder sister Aivy Mondal and sister-in-law Hridita Lasker Ritu for their support and inspiration in my bad times.

I would like to convey my greatest love and gratitude to my beloved wife, Eshita Lasker, and my lovely son, Barnabh. They are the love and inspiration of my life. Eshita always stayed with me in ease and hardship, inspired me constantly, and took away all stresses that made my work so easy. Barnabh helped me see life in a new light with his heavenly innocence, unforgettable smiles, and constant babbling.

Thanks to all of the members of the Software Research Lab with whom I have had the opportunity to grow as a researcher. In particular, I would like to thank Dr. Manishankar Mondal, C M Khaled Saifullah, Farouq Al. Omari, Shamima Yeasmin, Judith Islam, Zonayed Ahmed, Muhammad Mainul Hossain, Kawser Wazed Nafi, Amit Kumar Mondal, Tonny Kar, Golam Mostaeen, Rayhan Ferdous, Debasish Chakroborti, Hamid Khodabandehloo, Md Nadim, Avijit Bhattacharjee, Naz Zarreen Oishie, Abdul Awal, Md. Shamimur Rahman, Daniel Abediny, and Sristy Sumana Nath.

I am thankful to the Department of Computer Science of the University of Saskatchewan for their generous financial assistance through scholarships, awards, and bursaries that helped me to focus more deeply on my thesis work.

I would like to thank all of my friends and other staff members of the Department of Computer Science who have helped me to reach this stage. In particular, I would like to thank Gwen Lancaster, Greg Oster, Jeff Long, Cary Bernath, Smit Choksi, Sophie Findlay, Shakiba Jalal, Heather Webb, and James Ko.

I dedicate this thesis to my father, *Tarak Chandra Mondal*, and my school teacher, *Suresh Chandra Biswas*. Their priceless love, courage, and inspiration give me strength and hope to be a well-educated and a good human being. They both left the world prematurely, but their ideology still encourages me to serve humanity, and their directions help me to do something creative.

Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
API	Application Programming Interface
ARI	Automated Reading Index
ASCII	American Standard Code for Information Interchange
AST	Abstract Syntax Tree
CART	Classification and Regression Trees
CLU	Coleman Liau Index
CRUSE	Code Reusability
CUA	Code Understandability
DF	Degrees of Freedom
DS	Discouraged Sample
DT	Decision Tree
FDR	False Discovery Rate
GFI	Gunning Fox Index
GNB	Gaussian Naive Bayes
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IAC	inaccurate Claim
IDE	Integrated Development Environment
IDEF	Ill-defined Issue
KNN	K-Nearest Neighbor
ME	Metric Entropy
ML	Machine Learning
MWW	Mann-Whitney-Wilcoxon
NB	Naive Bayes
PS	Promoted Sample
Q&A	Question and Answer
RAM	Random Access Memory
RF	Random Forest
RIX	Readability Index
RMJM	Reproducible with Major Modification
RMM	Reproducible with Minor Modification
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
RPS	Random Promoted Sample
RSAQ	Randomly Sampled Answered Questions
RWM	Reproducible without Modification
SD	Standard Deviation
SMOG	Simple Measure of Gobbledygook
SO	Stack Overflow
SP	Sentiment Polarity
SQL	Structured Query Language
TAQ	Total Answered Questions
TCC	Text-code Correlation
TCR	Text-code Ratio
TE	Topic Entropy
TPS	Total Promoted Sample
TR	Text Readability
UAQ	Unanswered Questions

1 Introduction

1.1 Motivation

The online crowd-sourced developer forums such as Stack Overflow (SO) are becoming increasingly important and popular to software developers and practitioners due to the shortcomings in traditional and official knowledge resources [67, 99, 132]. SO has shaped the way developers search for solutions on the web for their programming related issues [93]. It has become a valuable knowledge base accumulating millions of questions and answers. It has more than 11 million registered users as of December 2019 [87]. Each user can ask questions or respond to questions by posting answers. The growth, and continued success of SO depends on two major factors, the participation of users and the quality of the shared knowledge [6, 57, 89]. However, despite having such a large and engaged community, about 29% (in the year 2018) of SO questions are not answered at all [87]. Besides, this percentage has been gradually increasing over the years [87]. Such a large number of unanswered questions might devalue the quality and purpose of the community-oriented knowledge base. Users of SO do not know either the questions will be answered or not during their submission. Detection of the potentially unanswered questions in advance during question submission could help one improve the question and thus receive the answers in time.

A large number of questions of SO discuss code issues (e.g., error and unexpected behavior) [9, 127]. Users often submit their questions with sample code examples to support their issue descriptions. Users of SO generally attempt to reproduce the programming issues discussed in the questions and then submit their solutions. Unfortunately, users could not always reproduce the programming issues, which might prevent these questions from getting prompt answers or answers at all. Users face several challenges in reproducing the issues reported at SO questions. Exploration of these challenges guides the users to include appropriate code segments with their questions that could help them to be answered and resolved.

SO also allows the users to improve the questions and answers by editing. In particular, editing helps to keep questions and answers clear, relevant and up-to-date by fixing grammar and spelling mistakes, clarifying the meaning, and adding related resources or hyperlinks. Li et al. [58] found that such collaborative editing increases the possibility of getting answers and also increase the score of questions and answers. However, such edits can be rejected due to the undesired modification of questions and answers. Therefore, it is important to understand why a suggested edit is being rejected and whether such edits may be prone to underlying ambiguities due to the lack of proper guidelines or proliferation of subjective bias.

While analyzing the questions, answers and their quality, we see several counter-intuitive findings on the reliability of the evaluation mechanism of SO. For example, about 48% of the questions that received working solutions have negative scores, about 18% of the accepted answers (i.e., verified solutions) also fail to receive the maximum votes among all answers in a Q&A thread [87]. The users of SO are also often disappointed by the evaluation against their questions and answers. All these findings above suggest the highly subjective nature of the votes cast by the users in SO. Furthermore, several previous studies also show that the incentive systems of Q&A sites may always not positively drive by certain users. For example, opportunistic users (i.e., users greedy to increase reputation in any way) may find loopholes, and aggressively game the system for their advantage [47, 52, 134]. To ensure a non-biased, reliable quality assessment mechanism of SO, a verification of the subjective evaluation is thus highly warranted.

1.2 Research Problem

A number of existing studies [5, 20, 102] are conducted to investigate the unanswered questions of SO. Asaduzzaman et al. [5] qualitatively analyze a total of 400 questions of SO and then report 13 factors (e.g., proprietary technology, irregular users) that might explain why the questions remain unanswered. However, many of their reported characteristics (e.g., impatient users) are hard to measure in practice. Saha et al. [102] introduce machine learning models to classify answered and unanswered questions. Unfortunately, the majority of their strong features (e.g., view count, score) are not available during the submission of a question. Thus, their model might not be able to predict the unanswered questions reliably during their submission. Calefato et al. [20] study such factors that might increase the chance of getting accepted answers to the SO questions. They suggest that high-quality presentation (e.g., body length), positive sentiment, question submission in weekdays, and high user reputation might improve the chance of getting answers. However, when they were analyzed none of these factors (except user reputation) was found powerful to reliably predict the unanswered questions during their submission. In short, the prediction of the unanswered questions is an open research problem that warrants further investigation.

Several existing studies [46, 147] investigate the usability and executability of the code segments posted on Q&A sites. Yang et al. [147] extract 914,974 code segments from the accepted answers of SO and analyze their parsability and compilability. However, their analysis was completely automatic, and they did not address the challenges of issue reproducibility. Horton and Parnin [46] analyze the executability of the Python code segments found on the GitHub Gist system. They identify several flaws (e.g., import error, syntax errors, indentation error) that prevent the execution of such code segments. However, a simple execution success of the code does not necessarily guarantee the reproduction of the issues discussed at SO questions. Thus, their approach also fails to address the reproducibility challenges that we are dealing with. Thus, there is a marked lack of research that investigates the challenges of issue reproducibility (of SO questions) and how reproducibility of issues affect the quality of SO questions.

Quality improvement of questions and answers posted in SO by editing were subject to a recent study by Wang et al. [134]. According to their investigation, a suggested edit was rejected for 12 reasons such as incorrect code change, undesired code/text formatting, and so on. Unfortunately, undesired code/text formatting could be prone to subjective bias. For example, two edits that suggest putting similar code terms (e.g., API names) into a code block $< code >< /code >$. However, the first suggested edit is approved, while the second one was rejected. Such observation leads to the question of how users are reviewing suggested edits and whether their judgment is clouded by individual bias. Lack of rules coupled with individual subjective bias could create ambiguity between the content owner and the users who suggested the edit. Such ambiguity may create confusion, frustrate the users, and dissuade the users from further contributing upon a rejection. Besides investigating rejection reasons for edits, it is thus necessary to understand the specific ambiguities that may be present in the rollback edits and use that to guide both forum designers and users. We are not aware of any previous study that analyzed ambiguities in SO rollback edits.

Several existing studies [18, 30, 79, 126] investigate questions and answers of SO and attempt to assess their quality. Nasehi et al. [79] analyze the quality of code segments included in SO posts and study the aspects that make up a good code example. Treude and Robillard [126] investigate to what extent developers perceive such code examples as self-explanatory. The appropriateness of code explanation is further studied by Ercan et al. [30] using Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [36]. Calefato et al. [18] focuses on the four light-weight, user-oriented features (e.g., user reputation) to predict the high-quality answers from technical Q&A sites. Novielli et al. [80] argue that the emotional style of a technical question could be a quality aspect that might influence the answering time of the question. Thus, the above studies simply rely on light-weight, ad-hoc attributes to estimate a post's quality. A few other studies [29, 93, 94] attempt to separate up-voted questions from down-voted questions using both the subjective (e.g., user reputation) and objective (e.g., text readability) metrics. While our work overlaps with them in terms of methodology, our research goals are different. In particular, we attempt to verify whether the subjective evaluation makes sense and the high-scored (hereby, promoted) questions on SO are actually preferable to the low-scored (hereby, discouraged) ones in terms of different objective quality metrics.

Thus the research problems we attempt to solve in this thesis have effective and diversified use to improve both the quality of content and assessment mechanism of SO.

1.3 Addressing Research Problems

The previous section briefly describes the systematic investigation that helps to identify the research problems on the quality of SO posts, their assessment and editing system, and also provides hints of their solutions. The following four studies have been conducted in total to address the above-mentioned problems.

- **Study 1:** Early Detection and Guidelines to Improve Unanswered Questions on SO

- **Study 2:** Can Issues Reported at SO Questions be Reproduced? An Exploratory Study

- **Study 3:** Rollback Edit Reasons and Ambiguities in SO
- **Study 4:** Do Subjectivity and Objectivity Always Agree? A Case Study with SO Questions

The following subsections briefly describe each of these studies.

1.3.1 Study 1: Early Detection and Guidelines to Improve Unanswered Questions on SO

In this study, we compare unanswered and answered questions quantitatively and qualitatively by analyzing a total of 4.8 million questions from SO. We find that topics discussed in the question, the experience of the question submitter, and readability of question texts could often determine whether a question would be answered or not. Our qualitative study also reveals several other non-trivial factors that not only explain (partially) why the questions remain unanswered but also guide the novice users to improve their questions. We develop four machine learning models to predict the unanswered questions during their submission. According to the experiments, our models predict the unanswered questions with a maximum of about 79% accuracy and significantly outperform the state-of-the-art prediction models.

1.3.2 Study 2: Can Issues Reported at Stack Overflow Questions be Reproduced? An Exploratory Study

In this study, we report an exploratory study on the reproducibility of the issues discussed in 400 questions of SO. In particular, we parse, compile, execute and even carefully examine the code segments from these questions, spent a total of 200 man-hours, and then attempt to reproduce their programming issues. The outcomes of our study are two-fold. First, we find that 68% of the code segments require minor and major modifications to reproduce the issues reported by the developers. On the contrary, 22% code segments completely fail to reproduce the issues. We also carefully investigate why these issues could not be reproduced and then provide evidence-based guidelines for writing effective code examples for SO questions. Second, we investigate the correlation between issue reproducibility status (of questions) and corresponding answer meta-data such as the presence of an accepted answer. According to our analysis, a question with reproducible issues has at least three times higher chance of receiving an accepted answer than the question with irreproducible issues.

1.3.3 Study 3: Rollback Edit Reasons and Ambiguities in Stack Overflow

Here, we offer, for the first time, a comprehensive overview of reasons and ambiguities in the SO rollback edits. Through an exhaustive qualitative analysis of 1,532 SO rollback and approved edits, we identify 14 reasons for rollback edits in SO and eight types of ambiguities that are often present in those edits. To analyze the prevalence of the ambiguities in SO, we develop a suite of algorithms to detect the ambiguities automatically. We ran our algorithms on around 102K SO rollback edits. We observe that presentation ambiguity is the most prevalent which arises due to confusion among developers on how to present contents (e.g., code terms). We also see that temporal ambiguities are rapidly increasing in SO, which arises when multiple already approved edits to a post got rejected due to a newer suggested edits - which is then rejected, rendering the valuable contributions of the approved edits useless. Our findings offer both qualitative and quantitative evidence on how the editing mechanism in SO should be improved to promote better content quality.

1.3.4 Study 4: Do Subjectivity and Objectivity Always Agree? A Case Study with Stack Overflow Questions

In this study, we compare the subjective assessment of questions with their objective assessment using 2.5 million questions and ten text analysis metrics. According to our investigation, (1) four objective metrics agree, (2) two metrics do not agree, and (3) one metric either agrees or disagrees with the subjective evaluation. We develop machine learning models to classify the *promoted* and *discouraged* questions, and also to predict them during their submission time. Our models outperform the state-of-the-art models with a maximum of about 76%–87% accuracy.

1.4 Related Publications

Below is the list of publications and the works that are prepared for submission (with collaborator) from this thesis.

- Saikat Mondal, Mohammad Masudur Rahman, Chanchal K. Roy. Can Issues Reported at Stack Overflow Questions be Reproduced? An Exploratory Study. *International Conference on Mining Software Repositories (MSR)*, 2019 pp. 479-489.

- Saikat Mondal, Gias Uddin, Chanchal K. Roy. Rollback Edit Reasons and Ambiguities in Stack Overflow. *International Conference on Automated Software Engineering (ASE)*, 2020. (under review).

- Saikat Mondal, C M Khaled Saifullah, Avijit Bhattacharjee, Mohammad Masudur Rahman and Chanchal K. Roy. Early Detection and Guidelines to Improve Unanswered Questions on Stack Overflow. *Journal of Systems and Software (JSS)* (to be submitted)

- Saikat Mondal, Mohammad Masudur Rahman, Chanchal K. Roy. Do Subjectivity and Objectivity Always Agree? A Case Study with Stack Overflow Questions. *Journal of Systems and Software (JSS)* (Under revise and resubmit)

1.5 Outline of the Thesis

The thesis contains seven chapters in total. In order to evaluate the quality of both the content and assessment mechanism of SO, we conduct four independent but interrelated studies and this section outlines different chapters of the thesis.

- Chapter 2 discusses the background concepts of this thesis such as Quality Assessment Mechanism of Posts of SO, edit system of SO, machine learning algorithms (e.g., random forest), Recall-Oriented Understudy for Gisting Evaluation (ROUGE) and so on.
- Chapter 3 discusses the first study that proposes an early detection technique of unanswered questions of SO and also guides users to improve their questions.
- Chapter 4 discusses the challenges to reproduce the issues reported in the questions of SO using the code segments included in the questions, and also investigates the relationship between the reproducibility status issues and answer meta-data.
- Chapter 5 discusses the rollback edit reasons and ambiguities in SO and also proposes algorithms to detect ambiguities automatically.
- Chapter 6 compares the subjective assessment of SO questions with their objective assessment using millions of questions and text analysis metrics and proposes machine learning techniques to classify promoted (i.e., high-quality) and discouraged (i.e., low-quality) questions.
- Chapter 7 concludes the thesis with a list of directions for future works.

2 Background

In this chapter, we provide a short discussion of the background and technical preliminaries to follow the remaining of the thesis. Section 2.1 discusses the question answering technique of SO and the different components of questions, Section 2.2 and Section 2.3 discuss the quality assessment and editing mechanism of SO, Section 2.4 describes the machine learning algorithms (e.g., decision trees, random forest) that employed in this thesis, Section 2.6 discusses the statistical significance tests (e.g., MWW test), Section 2.7 defines ROUGE that determines the quality of a text summary by comparing it to the summaries created by humans, and finally, Section 2.8 summarizes the chapter.

2.1 Question Answering on Stack Overflow

SO is an online community-based platform for question-answering in the domain of computer programming [149]. Users can post their questions about programming that are tightly focused on a specific problem [143]. A question has three components namely title, body, and tags. The title summarizes the problem whereas tags represent the topics related to the discussed problems. The body contains the textual description of the problems. Users often include code segments inside the body to support the explanation of the problem. Once a question is submitted by a user, others (i.e., answerers) can respond to this question by posting answers. When a question asker is satisfied with an answer, the asker can select the answer as the accepted answer among all the posted answers.

2.2 Quality Assessment Mechanism of Posts of Stack Overflow

The quality of the questions and answers in SO is subjectively assessed by the users through a voting system. Users can cast either up votes or down votes based on the quality of the questions or answers. Generally, high-quality posts (i.e., questions or answers) are voted up, on the contrary, low quality (e.g., unclear, vague, off-topic) posts are likely to be downvoted. A reputation system has been implemented on SO to measure and encourage the contributions of users to the community [149]. There are many ways for users to earn and lose reputation points. For instance, an answer or a question can be upvoted by other users; as a result, the answerer or question submitter gains 10 reputation points as a reward [86]. On the other hand, when an answer or a question is being downvoted, the answerer or the question submitter loses 2 reputation points [86].

2.3 Edit System in Stack Overflow

SO encourages users to edit the questions and answers to improve their quality. Conceptually, for a given content C shared in SO, the edit system S is composed of the five major entities: $S = \{U, C, E, P, D\}$, where (1) U is a set of users, (2) C is the content, (3) E is a list of suggested edits to the content, (4) P is the process of reviewing the suggestions, and (5) D is a decision that can be made on a suggested edit.

User U denotes both content owner U_C and other users offering edit suggestions U_E, i.e., $U = \{U_C, U_E\}$. Content C denotes a SO post (i.e., question or answer), $C = \{C_A, C_Q\}$. Depending on the type of participating user, the process P can of two types, normal P_N and expedited P_X (explained below). For a suggested edit, a decision can be either approve or reject, i.e., $D = \{D_A, D_R\}$.

Each suggested edit creates a new updated version of the current version of the content. Suppose the versions are denoted by $V = \{V_1, V_2, ..., V_n\}$. Upon an approval of a suggested edit on a version V_i, the edited version is identified by a new version V_{i+1} and the new version becomes current state of the content that is visible to everyone. Upon a rejection of a suggested edit on the version V_j, the content owner can decide to keep the current state of the content as V_j or rollback to a previous version V_i, where $i < j$. The rolled back version then becomes the current state of the content. The normal and expedited edit review processes are applied based on the reputation score of the user who suggested an edit. In a normal edit review process P_N, suggested edits of the users who have less than 2K reputation are placed in a review queue. In an expedited edit review process, P_X, users with more than 2K reputation can edit questions and answers without going through the review queue.

2.4 Machine Learning Algorithms

Machine learning (ML) is the systematic study of algorithms and statistical models that computer systems employ to perform a specific task [139]. It is about making computers modify or adapt their actions so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones [71]. In particular, machine learning algorithms build a mathematical model based on sample data, known as *training data*, to make predictions or decisions [10]. It does not use explicit instructions rather it relies on patterns and inference instead. It is regarded as a subset of artificial intelligence. Machine learning algorithms are used in a wide variety of applications, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task. Machine learning can be classified as follows.

Supervised learning. A training set of examples with the correct responses is given and, based on this training set with responses, the machine learning algorithm generalizes to respond correctly to all possible inputs [71]. This is also called learning from exemplars.

Unsupervised learning. Unlike supervised learning, correct responses are not provided, but instead, the algorithm attempts to identify similarities between the inputs so that inputs that have something in common are categorized together [71]. The statistical approach to unsupervised learning is known as density estimation.

Reinforcement learning. This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is incorrect but does not get told how to correct it [71]. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometimes called learning with a critic because of this monitor that scores the answer but does not suggest improvements [71].

In our thesis, we employ supervised machine learning algorithms to predict unanswered and discouraged (i.e., low quality) questions of SO. The algorithms are discussed as follows.

2.4.1 Naive Bayes (NB)

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting. It can also be represented using a very simple Bayesian network. Bayes theorem can be rewritten as –

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \tag{2.1}$$

The variable y is the class variable which represents the question category. Variable X represents the features. X is given as –

$$X = (x_1, x_2, x_3, \cdots, x_{10}) \tag{2.2}$$

Hence, $x_1, x_2, x_3, \cdots, x_{10}$ represent the features. By substituting for X and expanding using the chain rule we get –

$$P(y|x_1, x_2, ..., x_{10}) = P(x_1|y)P(x_2|y) \cdots P(x_{10}|y) \tag{2.3}$$

Now, we can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change. Therefore, the denominator can be removed and a proportionality can be introduced.

$$y = \arg\max{}_y P(y) \prod_{i=1}^{10} P(x_i|y) \tag{2.4}$$

Using the above function, we obtain the class, given the predictors. Since some of our features have continuous values we use *Gaussian Naive Bayes (GNB)*. Since the way the values are present in the dataset changes, the formula for conditional probability changes to –

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}) \tag{2.5}$$

Here, μ_y is the mean of the values in X associated with class y, and σ_y^2 is the variance of the values in X associated with class y. The major advantage of the naive Bayes classifier is its short computational time for training. In addition, since the model has the form of a product, it can be converted into a sum through the use of logarithms with significant consequent computational advantages.

2.4.2 Artificial Neural Networks (ANN)

Perceptrons can only classify linearly separable sets of instances. If a straight line or plane can be drawn to separate the input instances into their correct categories, input instances are linearly separable and the perception will find the solution. If the instances are not linearly separable learning will never reach a point where all instances are classified properly. Artificial Neural Networks (Multi-layered Perceptrons) have been created to try to solve this problem. A multi-layer neural network consists of a large number of units (i.e., neurons) joined together in a pattern of connections. Units in a net are usually segregated into three classes: input units, which receive information to be processed; output units, where the results of the processing are found; and units in between known as hidden units. Figure 2.1 shows the basic architecture of a Artificial Neural Network. First, the network is trained on a set of paired data to determine the input-output mapping. The weights of the connections between neurons are then fixed and the network is used to determine the classifications of a new set of data.

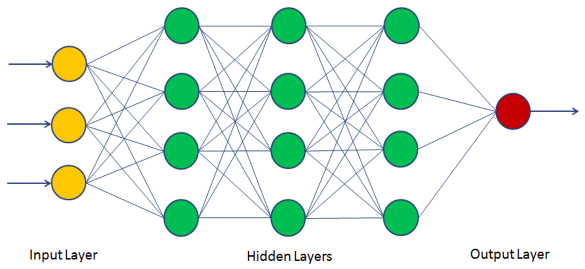


Figure 2.1: Architecture of ANN

During classification, the signal at the input units propagates all the way through the net to determine the activation values at all the output units. Each input unit has an activation value that represents some feature external to the net. Then, every input unit sends its activation value to each of the hidden units to which it is connected. Each of these hidden units calculates its own activation value and this signal are then passed on to output units. The activation value for each receiving unit is calculated according to a simple activation function. The function sums together the contributions of all sending units, where the contribution of a unit is defined as the weight of the connection between the sending and receiving units multiplied by the sending unit's activation value. This sum is usually then further modified, for example, by adjusting the activation sum to a value between 0 and 1 and/or by setting the activation value to zero unless a threshold level for that sum is reached. Generally, properly determining the size of the hidden layer is a problem, because an underestimate of the number of neurons can lead to poor approximation and generalization capabilities, while excessive nodes can result in overfitting and eventually make the search for the global optimum more difficult. Model overfitting means a model that learns the training dataset too well, performing well on the training dataset but does not perform well on a testing dataset.

ANN depends upon three fundamental aspects, input and activation functions of the unit, network architecture and the weight of each input connection. Given that the first two aspects are fixed, the behavior of the ANN is defined by the current values of the weights. The weights of the net to be trained are initially set to random values, and then instances of the training set are repeatedly exposed to the net. The values for the input of an instance are placed on the input units and the output of the net is compared with the desired output for this instance. Then, all the weights in the net are adjusted slightly in the direction that would bring the output values of the net closer to the values for the desired output.

2.4.3 Decision Trees (DT)

Decision trees are widely used in machine learning for classification tasks. They mainly split train sets according to some heuristics. Intermediate nodes represent conditions for splitting train set. Most of the commonly used decision trees like ID3, C4.5 and CART mainly apply different heuristics like entropy, information gain, gain ratio, gini index to decide which feature to be used for next split of the train set. ID3 can only be used when features are categorical. C4.5 and CART are extensions of ID3 which can work with features of both categorical and continuous data. Since our features have both continuous and Boolean values. So, CART or C4.5 are most suitable to use for above mentioned dataset.

$$Entropy(S) = \sum_{i=1}^{n} -P_{C_i} * \log_2(P_{C_i}) \tag{2.6}$$

$$Gain(S, A) = Entropy(S) - \sum_{i=1}^{m} P_{A_i} * Entropy(S_{A_i}) \tag{2.7}$$

Here, S is sample, P_{C_i} is the frequentist probability of a sample/class C_i, A is set of features, n is number of classes, m is categories for nominal value and ranges for continuous value. Entropy is the measure of uniformity. The more train set, S is balanced with classes, the more the entropy will be. For each attribute, Gain is calculated and then the most valued attribute is used to split set S. This process continues until we find all split resulting same class subset of S.

2.4.4 Random Forest (RF)

Random Forest proposed by Breiman [14] is an ensemble of decision trees. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner. In recent years, ensembles of classifiers have arisen as a possible solution even to the class imbalance problem [65]. Ensemble learners, also known as multiple classifier systems, try to improve the performance of single classifiers by inducing several classifiers and combining them to obtain a new classifier that outperforms every one of them

[92]. Random Forest is highly scalable to any number of dimensions and has generally quite acceptable performance [1]. In addition, it is very user-friendly in the sense that it has only two parameters (the number of variables in the random subset at each node and the number of trees in the forest) and is usually not very sensitive to their values [60]. The reasons behind the popularity of this algorithms are as follows.

- Random Forest deals with bagging of classification trees. Bagging (or bootstrap aggregating) is based on training many classifiers on boot-strapped samples from a training set, which has been shown to reduce the variance of the classification [37]. The usage of bagging enhances accuracy when random features are present and gives ongoing estimates of the generalization error of the combined ensemble of trees, as well as estimates for the strength and correlation [14].
- This algorithm runs efficiently on large datasets and for large data it creates highly accurate predictions. Our total dataset is pretty large as mentioned above.
- Uses of multiple trees reduce the risk of overfitting the data. Overfitting is a modeling error which occurs when a function is too closely fit to a limited set of data points.
- The random forest algorithm can be used for feature engineering i.e., identifying the most important features out of the available features from the training dataset. As we are working on 12 features, we need to understand from our data whether all the features are helpful for our prediction modeling.

In bagging, Random Forest generates successive trees that do not depend on earlier trees, each is independently constructed using a bootstrap sample of the data set [14]. At the end, a simple majority vote is taken for prediction [60]. The Random Forest approach introduces more randomness and diversity by applying the bagging method to the feature space. That is, instead of searching greedily for the best predictors to create branches, it randomly samples elements of the predictor space, thus adding more diversity and reducing the variance of the trees at the cost of equal or higher bias. This process is also known as *feature bagging* and it is this powerful method what leads to a more robust model [54]. In standard trees, each node is split using the best split among all variables. In a random forest, each node is split using the best among a subset of predictors randomly chosen at that node. This somewhat counter intuitive strategy turns out to perform very well compared to many other classifiers, including discriminant analysis, support vector machines and neural networks, and is robust against overfitting [14].

2.4.5 K-Nearest Neighbors(KNN)

K-Nearest Neighbor is a non-parametric [38] and lazy method used for classification and regression problems. Non- parametric means that it does not make any assumptions on the underlying data distribution. In other words, the model structure is determined from the data. Lazy means that it does not use the training data points to do any generalization. In other words, there is no explicit training phase, or it is very minimal. Thus, the training phase is fast. Here, k is the input and refers to the number of closest examples that the model will look for among the training data in the feature space. Output of the model represents the class to which the test data belongs to and it depends on the majority vote of its k-nearest neighbor [74]. Nearest Neighbor involves a training set of both positive and negative cases. A new sample is classified by calculating the distance to the nearest training case and the sign of that point then determines the classification of the sample [100]. The KNN classifier extends this idea by taking the k nearest points and assigning the sign of the majority. Larger k values help reduce the effects of noisy points within the training data set, and the choice of k is often performed through cross-validation [50].

This method is an instant-based learning algorithm that categorized objects based on closest feature space in the training set [41]. The feature space is partitioned into regions based on the category of the training set. A point in the feature space is assigned to a particular category if it is the most frequent category among the k nearest training data [50]. For finding closest similar points, we can find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance.

2.5 Metrics of Success

We evaluate the performance of our machine learning models using four appropriate performance metrics - *precision*, *recall*, *F1-score*, and *accuracy* [70]. We first generate confusion matrix that represents actual vs predicted number of samples for each category and then calculate the performance metrics. For describing confusion matrix, there are four terms. They are – (1) *True Positive (TP)* - the number of samples predicted as positive and are really positive, (2) *True Negative (TN)* - the number of samples predicted as negative and are actually negative, (3) *False Positive (FP)* - the number of samples predicted as positive, but actually negative, and (4) *False Negative (FN)* - the number of samples predicted as negative, but actually positive. Using these terms, we calculate the performance metrics as follows.

2.5.1 Precision

Precision is the ratio of accurately predicted positive sample and total predicted positive sample. So, for a class label X, Precision will be

$$Precision_X = \frac{TP_X}{TP_X + FP_X} \tag{2.8}$$

2.5.2 Recall

Recall is the ratio of accurately predicted positive sample and total positive sample in the data-set. So, for a class label X, Precision will be

$$Recall_X = \frac{TP_X}{TP_X + FN_X} \tag{2.9}$$

2.5.3 F1-score

F1-score describes how accurately and robustly a classifier predicts. It is harmonic mean of precision and recall. So, for a class label X, F1-score will be

$$F1_X = 2 * \frac{1}{\frac{1}{Precision_X} + \frac{1}{Recall_X}} \tag{2.10}$$

2.5.4 Accuracy

Accuracy means how many samples of overall prediction are correctly classified. For our work, accuracy for our model is the ratio of correctly predicted samples and total predicted samples.

$$Accuracy = \frac{\sum_i C_{i,i}}{\sum_{i,j} C_{i,j}} \tag{2.11}$$

2.6 Statistical Tests

The statistical significance tests that we use in this thesis are as follows.

2.6.1 Chi-Square Test of Independence

This statistical test is employed to determine whether there is a significant relationship between two nominal (i.e., categorical) variables [108, 109, 125]. The value of one variable does not change the probability distribution of another variable when the two categorical variables are independent. On the contrary, the distribution of one depends on the other when they are related to each other. The null hypothesis for this test is that the variables are independent, whereas the alternative hypothesis is that they are dependent. To measure the independence of two variables, we have to calculate the degrees of freedom, expected frequencies, test statistic, and the P-value associated with the test statistic using the given sample data as follows. Then we can interpret either the two variables are independent or not.

Degrees of Freedom (DF). The degrees of freedom is equal to –

$$DF = (r-1) \times (c-1) \tag{2.12}$$

where r represents the number of levels for one categorical variable, and c represents the number of levels for the other categorical variable.

Expected Frequencies. The expected frequency counts are computed separately for each level of one categorical variable at each level of the other categorical variable. Compute $r \times c$ expected frequencies, according to the following formula.

$$E_{r,c} = \frac{n_r \times n_c}{n} \tag{2.13}$$

where $E_{r,c}$ is the expected frequency count for level r of Variable A and level c of Variable B, n_r is the total number of sample observations at level r of Variable A, n_c is the total number of sample observations at level c of Variable B, and n is the total sample size.

Test Statistic. The test statistic is a chi-square random variable (χ^2) defined by the following equation.

$$\chi^2 = \sum \frac{(O_{r,c} - E_{r,c})^2}{E_{r,c}} \tag{2.14}$$

where $O_{r,c}$ is the observed frequency count at level r of Variable A and level c of Variable B, and $E_{r,c}$ is the expected frequency count at level r of Variable A and level c of Variable B.

P-value. The P-value is the probability of observing a sample statistic as extreme as the test statistic. Since the test statistic is a chi-square, we can use the Chi-Square Distribution Calculator to assess the probability associated with the test statistic.

Result Interpretation. If the sample findings are unlikely, given the null hypothesis, We can reject the null hypothesis. In particular, we can compare the P-value to the significance level, and reject the null hypothesis when the P-value is less than the significance level [125].

2.6.2 Mann-Whitney-Wilcoxon (MWW) Test

Mann–Whitney–Wilcoxon (MWW) test (a.k.a., Mann–Whitney U test, or Wilcoxon rank-sum test, or Wilcoxon–Mann–Whitney test) is a non-parametric test that can be used to investigate whether two independent samples come from the same population, or, alternatively, whether observations in one sample tend to be larger than observations in the other [140]. It compares two groups of data that are not normally distributed, such as those which are not measured exactly but rather as falling within certain limits [28]. Similar to other rank tests, the data in both groups being compared are initially arranged and listed in order of increasing value. Then each number in the two groups must receive a rank value. Beginning with the smallest number in either group (which is given a rank of 1.0), each number is assigned a rank. If there are duplicate numbers (i.e., ties), then each value of equal size will receive the median rank for the entire identically sized group. Thus if the lowest number appears twice, both figures receive the rank of 1.5. This, in turn, means that the ranks of 1.0 and 2.0 have been used and that the next highest number has a rank of 3.0. If the lowest number appears three times, then each is ranked as 2.0 and the next number has a rank of 4.0. Thus, each tied number gets a median rank. This process continues until all the numbers are ranked. Each of the two columns of ranks (one for each group) is totaled giving the sum of ranks for each group being compared.

There are different ways to evaluate the results. The easiest way advised by Wilcoxon, by which a test statistic TS is the sum of the ranks in the smaller group, i.e., the group with fewer observations (when groups are equally sized, either can be chosen) [28]. The statistic is evaluated in a special Mann-Whitney table. In other statistics, the sum of the ranks is TS and TL representing the smaller and larger rank sums, respectively. If the groups were the same the difference in rank sums should be small. A test statistic U is calculated as –

$$U = T_s - n_a \times \frac{(n_a + 1)}{2} \tag{2.15}$$

where n_a refers to the number in the group with the lower rank-sum. Consult a Mann-Whitney table which usually has the number of observations in both groups as entries. The target is to find the last probability column that does not contain the value of the statistic (U). There are different layouts of the tables but they only cover up to a total number of observations of about 20. If there are more observations, the rank-sum approaches a normal distribution and a z-value can be calculated.

2.6.3 Effect Size

Effect size is a quantitative measure of the magnitude of the experimenter effect [73, 141]. Effect size can be looked at when comparing any two groups to see how substantially different they are. Effect sizes either measure the sizes of associations between variables or the sizes of differences between group means [73]. Sullivan and Feinn [112] argue that p-value can inform the reader whether an effect exists, however, the p-value will not reveal the size of the effect. They suggest that in reporting and interpreting studies, both the substantive significance (i.e., effect size) and statistical significance (i.e., p-value) are essential results to be reported. While the sample size is sufficiently large, a statistical test usually almost always demonstrates a significant difference, even when the effect size is about zero. Thus, reporting only the significant p-value for analysis is not adequate for readers to fully understand the results [112].

Cohen's d is one of the most common ways to measure the effect size for the comparison between two means. To calculate the standardized mean difference between two groups, subtract the mean of one group from the other and divide the result by the standard deviation (SD) of the population from which the groups were sampled. We can thus calculate Cohen's d as follows.

$$Cohen's\ d = \frac{Mean_1 - Mean_2}{SD} \tag{2.16}$$

Cohen suggested that d=0.2 be considered a *small* effect size, 0.5 represents a *medium* effect size and 0.8 a *large* effect size. This means that if two groups' means don't differ by 0.2 standard deviations or more, the difference is trivial, even if it is statistically significant.

The Cliff's Delta statistic is another non-parametric effect size measure that quantifies the amount of difference between two groups of observations beyond p-values interpretation [69, 148]. Suppose that we have two samples $X = x_1, ..., x_m$ and $Y = y_1, ..., y_n$. Define the delta function as follows.

$$\delta(i,j) = \begin{cases} +1 & x_i > y_j \\ -1 & x_i < y_j \\ 0 & x_i = y_j \end{cases} \tag{2.17}$$

Cliff's delta effect size is then defined by

$$\delta = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\delta(i,j) \tag{2.18}$$

Here, values of δ near ± 1 mean that there is no overlap between the two samples, while values near zero indicate a lot of overlap between the two samples [148].

2.7 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

ROUGE includes measures to automatically determine the quality of a summary by comparing it to other ideal summaries created by humans [62]. In particular, the measures count the number of overlapping units such as n-gram, word sequences, and word pairs between the computer-generated summary to be evaluated and the ideal summaries created by humans. It includes several automatic evaluation methods such as – ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S. In this thesis, we use ROUGE-N that can be defined as follows.

ROUGE-N: N-gram Co-Occurrence Statistics. It is an n-gram recall between a candidate summary and a set of reference summaries [62]. ROUGE-N is computed as follows.

$$ROUGE-N = \frac{\sum_{S \epsilon \{ReferenceSummaries\} gram_n \epsilon S} \sum Count_{match}(gram_n)}{\sum_{S \epsilon \{ReferenceSummaries\} gram_n \epsilon S} \sum Count(gram_n)} \tag{2.19}$$

where n represents the length of the n-gram, $gram_n$, and $Count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries. ROUGE-N is a recall-related measure because the denominator of the equation is the total sum of the number of n-grams occurring at the reference summary side. When more references are added, the number of n-grams in the denominator of the ROUGE-N formula increases. The reason behind this is that there might exist multiple good summaries. Hence, the numerator sums overall reference summaries. This effectively provides more weight to matching n-grams occurring in multiple references. A candidate summary that contains words shared by more references is thus favored by the ROUGE-N measures [62]. This is very intuitive and reasonable because we normally prefer a candidate summary that is more similar to consensus among reference summaries.

2.8 Summary

In this chapter, we introduced different terminologies and background concepts that would help one to follow the remaining of the thesis. We discuss the question answering, quality assessment and editing mechanism of SO. We also discuss the different components of a question. We describe the machine learning algorithms (e.g., decision trees, random forest) that employed in this study. Then we discuss the statistical significance tests (e.g., MWW test). Finally, we define ROUGE, which is employed to measure the quality of a text summary by comparing it to the summaries created by humans.

3 Early Detection and Guidelines to Improve Unanswered Questions on SO

Existing attempts in detecting unanswered questions of SO primarily rely on such attributes (e.g., score, view count) of questions that are not available during the submission of a question. Detection of the potentially unanswered questions in advance during question submission could help one improve the question and thus receive the answers in time. In this chapter, we discuss our first study that proposes machine learning models to detect unanswered questions during their submission.

The rest of the chapter is organized as follows. Section 3.2 discusses the overall methodology of this study to answer the research questions. Section 3.3 reports the study findings. Section 3.3.1 discusses the comparison between unanswered and answered questions using several quantitative features, and Section 3.3.2 reports the findings from qualitative analysis and the agreement between quantitative and qualitative findings. Sections 3.3.3 describes the prediction models and their performance. Sections 3.4 reports the key findings, Sections 3.5 discusses the existing studies related to our research. Section 3.6 discusses threats to the validity of this study and finally, Section 3.7 summarizes the chapter.

3.1 Introduction

Software developers often search for solutions on the web for their programming problems. The advent of Q&A websites such as SO has shaped the way developers search for information on the web [93]. SO is the largest and most popular Q&A site that accumulates millions of programming related questions and answers. These questions and answers are consulted by a large technical community of more than eight million users (as of December 2018 [87]). Despite having such a large and engaged community, as already mentioned, about 29% of SO questions remain unanswered [87]. Besides, this percentage has been gradually increasing over the years (e.g., Fig. 3.1). Unanswered questions devalue a knowledge base in terms of quality and relevance. The lack of answers to the programming related questions also impedes the normal development progress. *Early detection* of a question that might not be answered helps one improve the question and thus reduces the answering time. Early detection means detection of the potentially unanswered questions in advance during a question submission.

Several existing studies [5, 20, 102] investigate the unanswered questions of SO. Asaduzzaman et al. [5] manually analyze 400 questions of SO and report 13 factors (e.g., proprietary technology, irregular users)

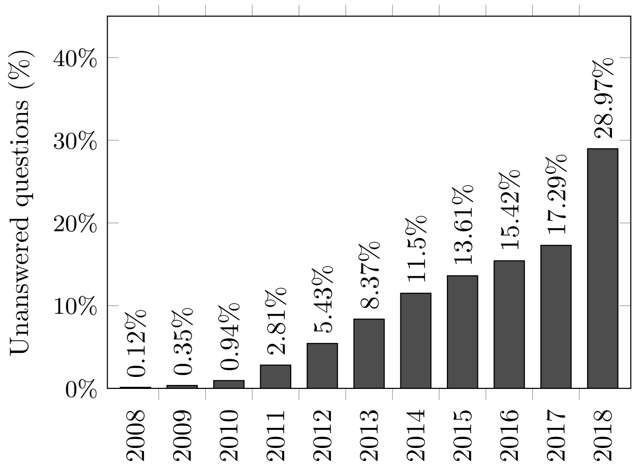


Figure 3.1: Percentage of the questions not answered in SO over eleven years [87].

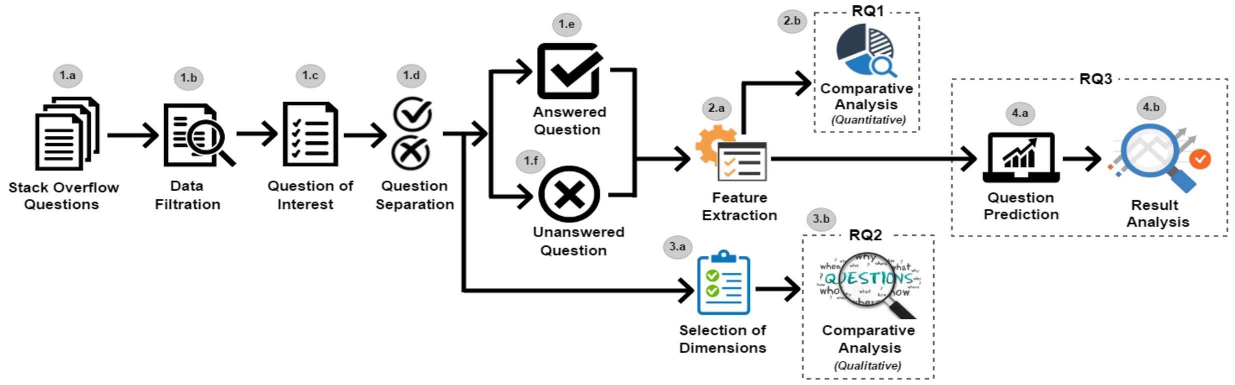


Figure 3.2: Schematic diagram of our conducted study.

that might explain why the questions remain unanswered. However, many of their reported characteristics (e.g., impatient users) are hard to measure in practice. Besides, the authors also do not examine whether their features could predict the unanswered questions or not. Saha et al. [102] introduce machine learning models to classify answered and unanswered questions. Unfortunately, majority of their strong features (e.g., view count, score) are not available during the submission of a question. Thus, their model might not be able to predict the unanswered questions reliably during their submission. Calefato et al. [20] study such factors that might increase the chance of getting accepted answers to the SO questions. They suggest that high quality presentation (e.g., body length), positive sentiment, question submission in weekdays, and high user reputation might improve the chance of getting answers. However, when they were analyzed (in section 3.3.3), none of these factors (except user reputation) was found significant enough to reliably predict the unanswered questions during their submission. In short, prediction of the unanswered questions is an open research problem that warrants further investigation.

In this study, we (1) present a comparative analysis between answered and unanswered questions of SO, and (2) provide four machine learning models to predict the unanswered questions. We conduct our study in three major phases. First, we analyze eight attributes of 4.8 million questions and their corresponding question submitters, and perform comparative analysis to determine how the unanswered questions differ from the answered questions. We find that the topics discussed in the question, experience of the question submitter, and readability of question texts could potentially determine whether a question would receive answers or not. Second, we manually investigate 400 questions (200 unanswered + 200 answered), and revisit the quantitative findings above to see whether the quantitative and qualitative findings support each other or not. Our qualitative findings not only confirm the quantitative findings above but also reveal several new insights. Third, we develop four machine learning models using the features of the questions that are available during their submission, and predict the unanswered questions. We choose four non-linear machine learning algorithms – Decision Trees (CART) [15], Random Forest (RF) [14], K-Nearest Neighbors (KNN) [3], and Artificial Neural Network (ANN) [136] – to train our prediction models. According to extensive experiments, our models predict the unanswered questions with a maximum accuracy of 79% which is highly promising. Comparison with two state-of-the-art models [20, 102] suggests that our models outperform them with statistically significant margins. We thus answer three research questions with our study as follows.

RQ1. How do the unanswered questions differ from the answered questions in terms of their texts and their submitters' activities? We conduct a comparative analysis between the unanswered and answered questions using eight quantitative metrics from the literature. We find that the topics of a question, the experience of a question submitter, and the readability of question texts could potentially determine whether a question would be answered or not. According to our analysis, questions related to programming IDEs (e.g., RStudio), libraries (e.g., TensorFlow), and frameworks (e.g., Angular6) often remain unanswered. Such questions are often discouraged at SO since they attract subjective opinions rather than authentic answers. Second, the users who have a higher reputation score are more likely to get answers to their questions. Third, easy-to-read questions are more likely to be answered.

RQ2. How do the unanswered and answered questions differ qualitatively? Do the qualitative findings agree with the quantitative findings? We conduct a manual investigation of 400 questions (200 unanswered + 200 answered) to see whether there is an agreement between the qualitative and quantitative findings. Besides supporting all the quantitative findings, our qualitative study also reveals several other non-trivial factors that partially explain why the questions remain unanswered. For example, questions that discuss outdated technology, multiple technical issues, or do not include appropriate code examples in their texts are more likely to be not answered. Our findings also align with the guidelines posted by SO about asking a good question [84]. However, their guidelines are theoretical. On the contrary, our findings and insights are backed up by empirical evidence that can explain why certain questions are likely to receive answers and the others are not. Moreover, our models can help the developers improve their questions with instant predictions on their quality. Several questions receive working solutions in their comment thread. Thus, although they seem to be unanswered, they are actually answered.

RQ3. Can we predict the unanswered questions of SO during their submission? We develop four machine learning models to predict the unanswered questions during their submission. Our models predict the unanswered questions with a maximum of about 79% accuracy which is significantly higher than that of the state-of-the-art models.

3.2 Study Methodology

In Fig. 5.3, we describe our overall methodology to answer the three research questions. We describe the steps below.

Step 1: Data Collection and Preprocessing. Table 3.1 shows the summary of our collected dataset. We collect SO questions using StackExchange Data API [87] (e.g., Fig. 5.3, Step 1.a). In particular, we collect questions related to four popular programming languages - C#, Java, JavaScript, and Python. We choose the questions that were submitted on the site in the year 2017 or earlier (e.g., Fig. 5.3, Step 1.b). We wanted to ensure enough time for each question to be assessed by the SO community. We get a total of 4,814,900 questions where 4,324,252 questions have one or more answers, and the remaining 500,648 questions are unanswered (e.g., Fig. 5.3, Steps 1.c – 1.f). To calculate user reputation, we collect the information of the SO users and the votes they received on their questions and answers. We also collect the data associated with SO tags assigned to the questions.

Step 2: Compare Unanswered and Answered Questions. We extract eight features (e.g., Text Readability, Topic Response Ratio, Topic Entropy) from both unanswered and answered questions (e.g., Fig. 5.3, Step 2.a), and conduct a comparative analysis (quantitative) between the attributes of these two categories (e.g., Fig. 5.3, Step 2.b).

Step 3: Agreement Analysis. We manually analyze 400 questions (200 unanswered + 200 answered) to investigate the qualitative difference between unanswered and answered questions (e.g., Fig. 5.3, Steps 3.a, 3.b). In particular, we attempt to determine whether the qualitative findings agree with the quantitative findings or not.

Step 4: Predict Unanswered Questions. We build four machine learning models to predict whether the SO questions might be answered or not (e.g., Fig. 5.3, Step 4.a). Finally, we analyze the results of our models and also compare with the state-of-the-art models (e.g., Fig. 5.3, Step 4.b).

Table 3.1: Summary of the study dataset

	C#	**Java**	**JavaScript**	**Python**	**Total**
Answered	1,034,516	1,182,921	1,340,016	766,799	4,324,252
Unanswered	113,903	143,757	158,344	84,644	500,648
Total	1,148,419	1,326,678	1,498,360	851,443	**4,814,900**

3.3 Study Findings

We ask three research questions in this study. In this section, we answer them carefully with the help of our empirical and qualitative findings as follows:

3.3.1 Answering RQ1 : Comparison between Unanswered and Answered Questions using Quantitative Features

In this section, we contrast between the unanswered and answered questions using eight metrics that capture different aspects (e.g., readability, topic) of a question. Each of them is discussed as follows.

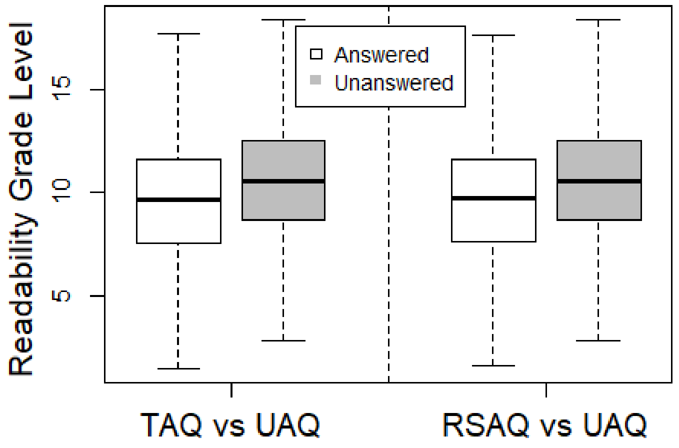


Figure 3.3: Text readability (**TAQ**=Total Answered Questions, **RSAQ**=Randomly Sampled Answered Questions, **UAQ**=Unanswered Questions).

(M1) Text Readability. Readability of a text document depends on the complexity of its vocabulary, syntax and the presentation style (e.g., line lengths, word lengths, sentence lengths, syllable counts) [138]. We extract the title and textual description from the body of each question to measure its text readability. The existing readability metrics are not well designed for handling the code or program elements within the texts [91]. We thus discard the code elements, code segments, and stack traces from the textual description using specialized HTML tags. For example, code elements are enclosed by <*code*> tag, code segments and stack traces are enclosed by <*code*> under <*pre*> tag. Unfortunately, some users do not use appropriate tags. We thus attempt to remove the code elements (e.g., method name) and stack traces from texts by employing appropriate regular expressions, as used by the existing literature [77]. We compute five popular readability metrics: *Automated Reading Index (ARI) [106], Coleman Liau Index [24], SMOG Grade [68], Gunning Fox Index [39], Readability Index (RIX) [4].* These metrics are widely used to estimate the comprehension difficulty of the English language texts [93]. First, we calculate the individual score (i.e., grade level) for each of the five metrics, and then determine the average readability of each question's text. Here, the lower the grade level, the easier the text is to read, and conversely, the higher the grade level, the more difficult the text is to read.

As shown in Fig. 3.3, we find a significant difference in the text readability between answered and unanswered questions. Answered questions have higher text readability (i.e., lower grade level) than that of the unanswered questions. Unanswered questions often contain long, multi-syllable words whereas the answered questions generally contain small, simple words. For example, according to our analysis, the average number of long (i.e., $length > 6$) and multi-syllable words per answered question are 17.8 and 27.8, whereas the same numbers for the unanswered question are 20.8 and 32.4 respectively. Since word length and syllable are important dimensions of the traditional readability tools [4, 24, 39, 68, 106], the low readability of the unanswered questions is pretty much explained. We also address the data imbalance issue during our comparative analysis. Since the number of answered questions is higher than that of the unanswered questions in our dataset, we randomly under-sample the answered questions to balance the dataset. Then we compare the readability scores between unanswered and answered questions again with the sampled data. Interestingly, we were able to reproduce the same finding as above. We also perform statistical significance tests such as Mann-Whitney-Wilcoxon and Cliff's Delta, and found statistical significance for all four programming languages (i.e., p-value = 0 <.05 and Cliff's |d| = 0.17 (*small*)). That is, the readability of the answered questions is significantly higher than that of the unanswered questions regardless of the programming languages. Since we perform multiple comparisons (e.g., comparisons of unanswered, total answered and random sampling answered) our result could be affected by the type I error in null hypothesis testing. In statistical hypothesis testing, a type I error is the rejection of a true null hypothesis [142]. To mitigate this problem, we control the false discovery rate (FDR) by adjusting the p-values based on the method of Benjamini and Yekutieli [8].

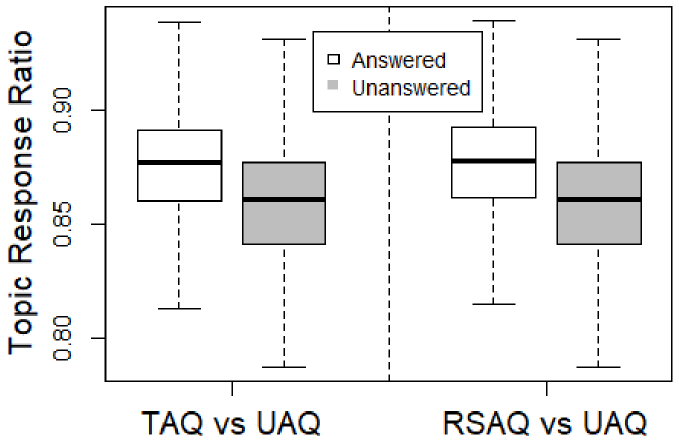


Figure 3.4: Topic response ratio (**TAQ**=Total Answered Questions, **RSAQ**=Randomly Sampled Answered Questions, **UAQ**=Unanswered Questions).

(M2) Topic Response Ratio. In SO, questions often might not receive answers due to their topical difficulty. Questions related to certain topics (e.g., software libraries, platform, IDE) usually get less attention than others (e.g., jquery-selectors, lisp) in SO. Questions on these topics might attract subjective opinions rather than the authentic answers. Thus, these questions are discouraged and often redirected to other

Table 3.2: Comparison between unanswered and answered questions using quantitative features

Feature	Answered Question			Unanswered Question			MWW Test (p-value)	Cliff's $\lvert d \rvert$
	Mean	Median	Std. Dev.	Mean	Median	Std. Dev.		
Text Readability	9.89	9.67	4.12	11.05	10.53	4.90	0	0.17 (*small*)
Topic Response Ratio	0.88	0.88	0.04	0.86	0.86	0.05	0	0.40 (*medium*)
Topic Entropy	2.63	2.66	0. 67	2.73	2.75	0.67	0	0.09 (*negligible*)
Metric Entropy	0.0068	0.0064	0.0026	0.0061	0.0058	0.0023	0	0.17 (*small*)
Average Terms Entropy	0.0004	0.0004	0.0001	0.0004	0.0003	0.0001	0	0.09 (*negligible*)
Text-code Ratio	1.35	0.46	4.00	1.17	0.36	3.85	0	0.09 (*negligible*)
User Reputation	62.97	22.0	131.73	39.16	10.0	110.76	0	0.24 (*small*)
Received Response Ratio	0.71	1.0	0.45	0.60	0.87	0.45	0	0.30 (*small*)

community websites (e.g., GitHub) for appropriate answers [82].

We first examine which topics are more likely to receive answers and which are not. In SO, tags often capture the topics of a submitted question. Each question can have at most five tags and must have at least one tag [150]. We first calculate the answering ratio of each topic, R_t as follows.

$$R_t = \frac{A_Q}{T_Q} \tag{3.1}$$

where A_Q and T_Q represent the number of answered questions and total questions associated with the topic respectively. Since each question can have multiple topics (tags), we then measure the topic response ratio (Q_{resp}) of each question as follows.

$$Q_{resp} = \frac{1}{N} \times \sum_{i=1}^{N} R_{t_i} \tag{3.2}$$

where N is the number of topics associated with a question and R_{t_i} denotes the response ratio of the i^{th} topic of the question. According to our investigation, as shown in Fig. 3.4, the topic response ratio of the unanswered questions is lower than that of the answered questions. It indicates that the topics of the unanswered questions are of little interest to the SO user community. It is also possible that there are not sufficient experts to answer the topics discussed in the unanswered questions. We also found this difference as statistically significant using Mann-Whitney-Wilcoxon and Cliff's delta statistical test (i.e., p-value = 0 <.05 and Cliff's d = 0.40 (*medium*)).

(M3) Topic Entropy. Single tags (e.g., java) assigned to a question often fail to explain the topics or subject matter of the question. On the contrary, multiple tags attached to a question could help the users better understand the topics and thus help them find the questions of their interest. We analyze the ambiguity of question topics and attempt to determine whether the unanswered questions use more ambiguous topics than those of the answered questions. In information theory, entropy is used as a measure of uncertainty of a random variable that takes up multiple values [97]. Similarly, we consider the probability of a question discussing a certain topic as a random variable, and determine topic entropy of each question. To measure the topic entropy of each question, we first calculate the prior probability (P_k) of each topic (i.e., tag) associated with the question at SO. P_k is defined as a ratio between the number of questions discussing a topic $k \in n$ and the number of questions discussing all the topics of SO. We then determine the topic entropy for each question as follows.

$$TE = -\frac{1}{n}\left[\sum_{k=1}^{n} P_k \times log(P_k)\right] \tag{3.3}$$

where n denotes the total number of topics of a question. Here, we calculate the average entropy (dividing the entropy by n) since the number of tags differ from question to question. If a question of SO uses one or more ambiguous topics, the entropy becomes higher.

We contrast between unanswered and answered questions in terms of their topic entropy. Table 3.2 summarizes the comparative analysis. We see that topic entropy of answered questions is significantly lower than that of the unanswered questions. Such a result indicates that the unanswered questions are more ambiguous than the answered questions, which possibly left them with no answers.

(M4) Metric Entropy and (M5) Average Terms Entropy. We use Metric Entropy and Average Terms Entropy to estimate the randomness of terms used in the textual part of the unanswered and answered questions. Metric Entropy is the Shannon entropy [123] divided by the character length of the question's text. It represents the randomness of the terms against a question [93]. Average Terms Entropy also estimates the randomness of each term for the question's text. However, in this case, the entropy of each term is calculated against all the questions on SO. We first calculate the entropy for each term in the SO data dump of December 2017 [32]. We then determine the equivalent entropy of each term for a question's text against all the questions. Finally, we calculate the Average Term Entropy for each question by summing each of the terms entropy for the question divide by the character length of the question's text. The entropy value describes the discriminating power of a term [93]. The lower the entropy of a question, the higher the use of uncommon terms in the question.

As shown in Table 3.2, we find a relatively lower metric and average terms entropy for the unanswered questions than that of the answered questions. That is, the unanswered questions use more uncommon terms in their texts than the answered questions do. It indicates that the unanswered questions might use inappropriate terms that do not describe the problem correctly. Moreover, they might touch such topics that often remain unanswered. According to our analysis, we found the differences are statistically significant (i.e., p = 0 <.05) using the Mann-Whitney-Wilcoxon test. However, the effect size is either small or negligible.

(M6) Text-code Ratio. The ratio between text and code of a question is often considered as an important dimension of question quality [110]. Large code segments with little explanation might make a question hard to comprehend. We thus wanted to find out whether the unanswered and answered question differ from each other in terms of their text-code ratio. We extract the code segments and texts from the body of a question. Then we calculate the text-code ratio by dividing the character length of the text with the length of the code segment. When the code segment is absent from a question, we assign a large number as the text-code ratio.

According to our analysis, the text-code ratio of the unanswered questions is lower than that of the answered questions (as shown in Table 3.2). That is, insufficient explanation of the code segment could be a potential factor that might explain why the questions remain unanswered. We also measure whether the difference of text-code ratio between unanswered and answered questions is statistically significant. From the Mann-Whitney-Wilcoxon test, we find significant p-value (p-value = 0 <.05). However, the effect size from the Cliff's Delta test was found negligible.

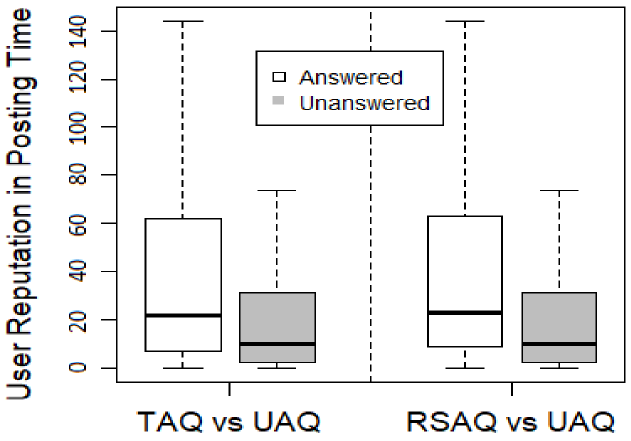


Figure 3.5: User reputation (**TAQ**=Total Answered Questions, **RSAQ**=Randomly Sampled Answered Questions, **UAQ**=Unanswered Questions).

(M7) User Reputation. The reputation system of SO is designed to incentivize contributions and to allow assessment from the users [18]. Asaduzzaman et al. [5] argue that a question submitted by a user with a higher reputation has a higher possibility of getting answers. We thus consider the users' reputation as a distinctive feature of SO questions. The official data dump only reports the latest reputation scores of the users, which might not be appropriate for our analysis. However, SO stores all the activities (e.g., votes, acceptances, bounties) of users with their dates. We thus use the snapshot of the activities of users to calculate the reputation during their question submission. For reputation calculation, we used a standard equation provided by the SO [33].

According to our analysis (Fig. 3.5), the reputation of the authors of unanswered questions is lower than that of the authors of answered questions. It confirms that the high reputation score increases the chance of getting answers. Using the Mann-Whitney-Wilcoxon test, we find a significant p-value (i.e., p-value = 0 <.05) although the effect size is small (i.e., Cliff's d = 0.24).

(M8) Received Response Ratio. The past question-answering history of a user might provide useful information as to whether a question submitted by the user will be answered or not [102]. Thus we investigate a user's question-answering statistics. In particular, we measure the percentage of answered questions (AQ_p) asked by users during their new question submission as follows.

$$AQ_p = \frac{\sum_{t<T} A_q}{\sum_{t<T} T_q} \times 100 \tag{3.4}$$

where T is the submission time of the question currently being processed, t denotes the submission time of the previous questions, A_q and T_q represent the total number of answered questions and asked questions respectively.

According to our analysis, a user with a higher percentage of receiving answers in the past has a higher chance of receiving answers in the future. As shown in Table 3.2, the authors of the answered questions received answers for about 71% of their submitted questions in the past. On the contrary, the same percentage is about 60% for the authors of the unanswered questions. The difference is statically significant (i.e., p-value = 0 <.05) with a small effect size (i.e., Cliff's d = 0.30).

3.3.2 Answering RQ2 : Agreement between Quantitative and Qualitative Findings

While our quantitative analysis provides enough evidence that unanswered questions are significantly different from the answered questions, we further conduct qualitative analysis to better understand their difference. In particular, we attempt to determine whether the qualitative findings agree with the quantitative findings from the above section (Sec. 6.6.6) or not.

Study Setup. In SO, the topics of a question are often conveyed through predefined tags. We first find the popular tags that were assigned to at least 5K questions of SO. Then we find the questions that use one of these tags. We count the number of unanswered and answered questions connected to each topic. We then measure the ratio of unanswered and answered questions against each given topic. We then find the top 10 topics that were unanswered (e.g., jupyter-notebook, electron, rstudio) and another top 10 that were frequently answered (e.g., scheme, sql-server-2005, asp.net-mvc-2). Finally, we randomly choose 20 questions for each of these 20 topics, manually analyze a total of 400 questions (statistically significant with a confidence level of 95% and confidence interval of 5% [12]), and derive several qualitative insights.

Manual Analysis. Two of the authors conduct a qualitative study to see the different quality aspects of unanswered questions, contrasting them with answered questions. We analyze 400 questions by spending a total of 40 man-hours. We first examine the randomly sampled questions with no particular viewpoints in mind. We then discuss together to share our understanding and set a few dimensions to proceed with our analysis. Next, the sampled data is presented to each author for manual analysis. The authors rank them with four major quality aspects from 0 (lowest) to 5 (highest). The aspects are – i) problem description and code explanation ii) topic assignment ii) question formatting and iv) appropriateness of the code segments included with questions. Besides the above aspects, we also investigate whether the question was received an answer in a comment, redirected to another site (e.g., GitHub), asked suggestions, raised multiple issues in a single question, and marked as a possible duplicate. During the manual investigation, we periodically sat

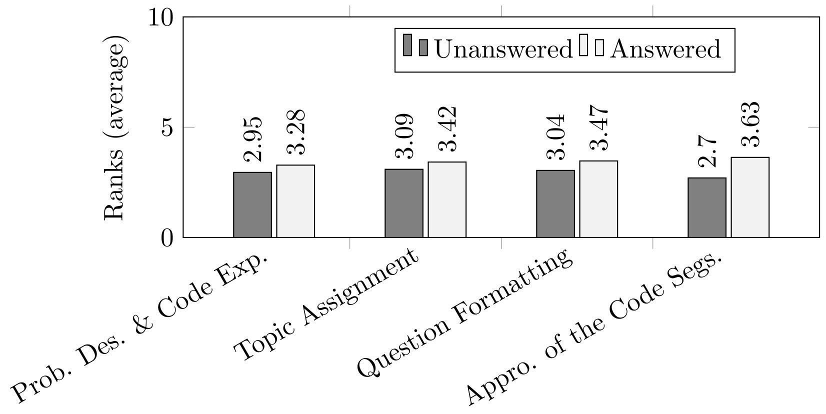


Figure 3.6: Comparison (qualitative) between unanswered and answered questions.

together and discussed the major inconsistencies in ranking until a consensus was reached. In particular, we resolved the inconsistencies by discussion when our ranking varied more than two in any aspect of a question.

Results. Fig. 3.6 shows the results of our manual analysis. For each of the four aspects, the average ranking for unanswered questions is lower than that of the answered questions. For example, the appropriateness of the code segments is 2.7 (on the scale of 5) for unanswered questions. Such rank is 3.63 for the answered questions. Our findings from the manual analysis are as follows.

• *Problem description and explanation of code segments*: We find that the problem description of the answered questions is complete, relevant, and easy to read. These questions include code segments when required and provide a useful explanation of the code elements so that users can comprehend them easily. When a question includes several code segments, we find a separate explanation for each of them. On the other hand, we find that the problem description is not sufficient and somewhat irrelevant in the unanswered questions. Questions without a clear problem statement are not useful to the readers. In the case of unanswered questions, we found long, redundant code segments that were not explained properly. They also do not provide any markers that could help one find the faulty part of the code. Sometimes the environment settings (e.g., operating system) and other technical details (e.g., software version) were found missing as well. In the quantitative analysis (section 6.6.6), we also find that the text-code ratio of the unanswered questions is lower than that of the answered ones. Thus, our manual investigation supports the quantitative difference in the text-code ratio between unanswered and answered questions.

• *Users experience and usage of appropriate terms.* We often see popular technical terms (e.g., GUI) in the texts of the answered questions which are familiar to the technical community. Such a use of popular words might improve the overall clarity of the question texts. On the contrary, we frequently notice the use of non-technical terms in the unanswered questions. This might occur due to the lack in experience (i.e., novice user) and domain expertise. In the quantitative analysis, we also find that (1) the submitters' reputation of the unanswered questions is lower than that of the submitters' of the answered questions on average, and (2) unanswered questions use more uncommon terms. Thus, our manual investigation agrees with the quantitative difference in the word uses between unanswered and answered questions.

• *Assignment of appropriate topics.* Tags assigned to the answered questions are precise and interesting enough to attract the experts of the domain. On the contrary, the tags assigned to the unanswered questions are generic (e.g., java). Our topic entropy shows that the tags of the unanswered questions are more generic, less precise than that of the answered questions. Users with low reputation (e.g., $reputation < 1500$) cannot add new tags to their questions [83] and are forced to use the existing generic or even potentially non-appropriate tags. Such a constraint might also prevent novice users of SO to assign proper tags to their questions.

3.3.3 Answering RQ3 : Construction of Prediction Models and Result Analysis

RQ1 and RQ2 investigate why the questions of SO remain unanswered using quantitative and qualitative approaches. Predicting a question (during its submission) that might not be answered could help one (1) improve the question and (2) thus receive the answers in time. We construct four machine learning models to predict whether a given question would be answered or not. In this section, we describe how we construct these models and evaluate their performance. We also compare our model performances with the baseline performances. Finally, we present a case study to demonstrate the generalizability of our models.

Algorithm Description. We use four algorithms for building prediction models – i) Decision Trees (CART) [15], ii) Random Forest (RF) [14], iii) K-Nearest Neighbors (KNN) [3], and iv) Artificial Neural Network (ANN) [136]. We choose these algorithms for two reasons. First, they are capable of building reliable models, even when the relationship between the predictors and class is nonlinear or complex. According to our comparative study, the relationship between question classes and their corresponding feature values might be complex. Second, these algorithms are widely used in the relevant studies [93, 97, 102]. Thus, we choose these four popular machine learning techniques that have different learning strategies to predict the unanswered questions. Moreover, they are non-parametric machine learning techniques. That is, they do not make any assumptions on the underlying data distribution.

Classification and Regression Tree (CART) and Random Forest (RF) are tree-based algorithms where the trees are constructed using predictors as non-leaf nodes and the classes as the leaf nodes. They show how individual features can affect the overall prediction. On the other hand, algorithms such as ANN and KNN combine all features (a.k.a., predictors) for determining the class label of an instance. ANN constructs a non-liner classification model during training, whereas KNN sorts the multi-dimensional feature space based on their classes.

Metrics of Success.

We evaluate our prediction models using four appropriate performance metrics - *precision*, *recall*, *F1-score*, and *classification accuracy.*

Model Configuration. We use a grid search algorithm, namely *GridSearchCV*, from the scikit-learn library [90] to select the best model configuration. GridSearchCV algorithm works by running and evaluating the model performance of all possible combinations of parameters. We also examine the accuracy of both train and test datasets and adjust the parameters so that models do not over-fit. We use the following parameter settings to configure our models.

ANN is configured using two hidden layers with each having eight activation units, logistic *Sigmoid function* as the activation function, *adam* as the weight optimizer, learning rate of 0.0001 and a batch size of 200. For KNN, we apply the K-Dimensional Tree (KDTree) algorithm to compute the nearest neighbors between 50 neighbors where, distance is used for weight function. For DT, Gini is used as the function to measure the quality of split and two is used as the depth of the tree. For RF, the number of trees is used 5 and Gini is used for splitting criteria.

Dataset Selection. We keep the training and testing data separate. We use the attributes of the questions submitted on SO in the year 2016 or earlier for training and questions submitted in the year 2017 for the testing purpose. We have several time-dependent features (e.g., user reputation). Thus, we make sure that we predict the unseen questions based on past questions.

Prediction of Unanswered Questions. To see how well the prediction models perform based on our features, we conduct an experiment with our models. Table 3.3 shows the experimental results of our models.

Table 3.3: Experimental Results

Technique	**Question**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
Decision Trees	Unanswered	72.57%	67.37%	69.87%	70.95%
	Answered	69.55%	74.54%	71.96%	
Random Forest	Unanswered	77.68%	80.83%	79.22%	78.80%
	Answered	80.02%	76.77%	78.36%	
ANN	Unanswered	58.59%	72.37%	64.75%	60.60%
	Answered	63.87%	48.84%	55.35%	
KNN	Unanswered	67.62%	67.46%	67.54%	67.57%
	Answered	67.54%	67.69%	67.62%	

We see that all four models can predict the question classes with 61%-79% accuracy. Our primary focus is to predict the unanswered questions. We thus analyze how the models perform to predict them. Random Forest performs the best among four models with precision 77.68% and recall 80.83%. Decision Tree is found as the second best with precision 72.57% and recall 67.37% followed by KNN with precision 67.62% and recall 67.46%, and ANN with precision 58.59% and recall 72.37%. One clear distinction can be drawn from the results that, individual feature processing algorithms such as Random Forest and Decision Tree are comparatively better suited for our dataset than the collective feature processing algorithms such as ANN and KNN. Moreover, Random Forest employs an ensemble learning, considers multiple trees to come to a decision and thus can avoid over-fitting and suited better for our dataset. Constructing multiple decision trees also helps the Random Forest model to capture non-linear relations between the predictors and classes more accurately for our problem. On the contrary, our dataset might not be well suited for ANN and KNN.

We investigate 40 cases (20 unanswered + 20 answered) where all of our models fail. Such insights might help and motivate future studies to improve models' performance. According to our investigation, our models misclassify the unanswered questions that are related to a few rising technologies (e.g., netflix zuul, libgdx). We see that the presentation quality, explanation of code segment, readability of text, and topic selection of these questions were reasonably well. Unfortunately, they might be remained unanswered due to the lack of experts. Our models often inaccurately predict the answered questions when they have long code with a more concise explanation. We use the text-code ratio instead of the length of the text to overcome such situations. However, our models fail to handle a few extreme cases [85]. Our analysis suggests that these questions received answers because either they included self-explanatory code segments or the code segments were capable of reproducing the issues reported at the questions [75].

Feature Strength Analysis. While answering RQ1, we showed that there are several attributes, whose values are different for unanswered and answered questions. Such findings indicate that these attributes might be the key estimators in predicting whether a question will be answered or not. A ranking of these attributes might help to select the top estimators in the prediction task. We thus use *ExtraTreesClassifier*, an inbuilt class of scikit-learn [90] to rank the features.

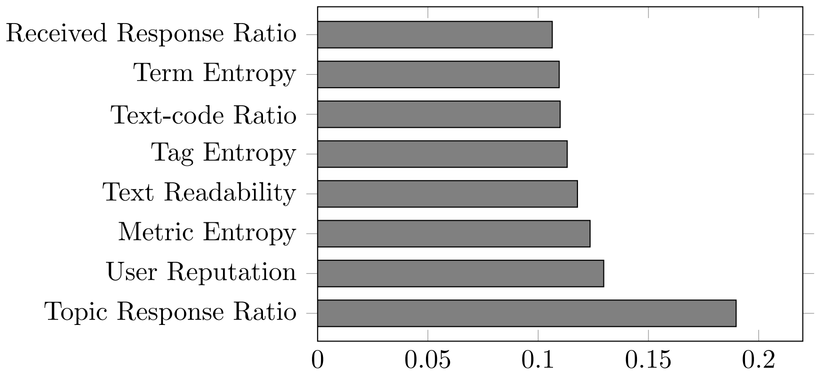
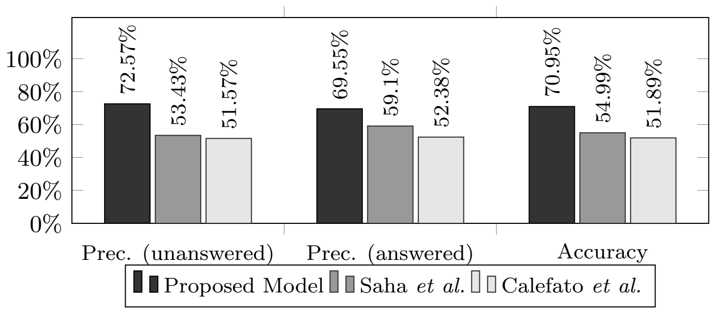


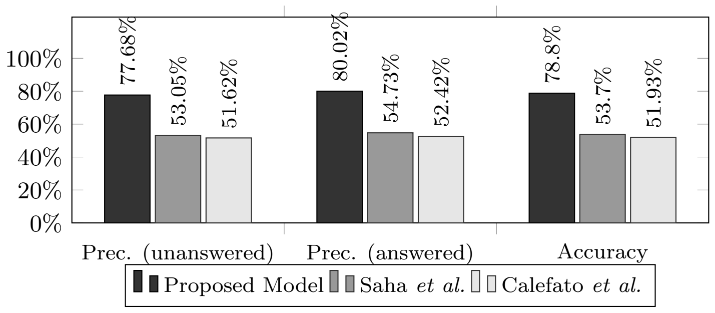
Figure 3.7: Feature Rank

As shown in Fig. 3.7, topic response ratio has the highest classification strength. It indicates that selection of appropriate tags (or topics) during question submission at SO is important. According to our investigation, topics related to software libraries, frameworks are less likely to be answered in SO. User reputation is the second most effective feature. That is, the trusted users by the SO community are more likely to get answers in response to their questions. Reputation also serves as a proxy to a user's skills or expertise in a particular domain. The next two important features are metric entropy and text readability. They suggest that the use of terms and reading ease of the question texts is important to receive answers. The use of ambiguous complex terms can prevent a question from being answered. The remaining four features have low impacts on receiving answers.

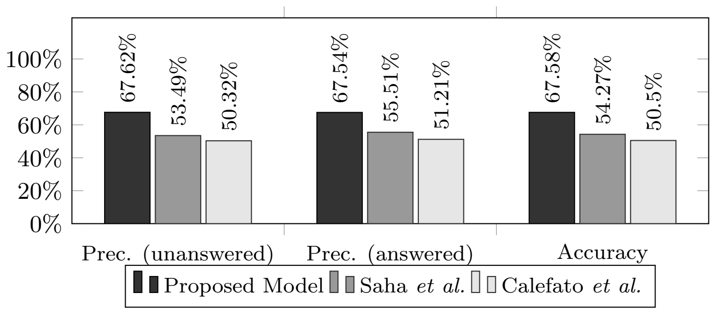
Comparison with Baseline Models. According to the results (Table 3.3), our proposed models able



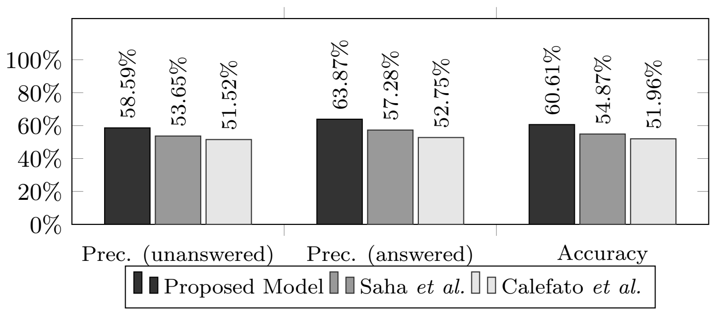
(a) Decision Trees



(b) Random Forest



(c) K-Nearest Neighbors



(d) Artificial Neural Network

Figure 3.8: The performance of different algorithms applied to the raw feature values.

to predict questions about 60–80% of the time correctly. However, we are interested to compare our models with the models studies previously. Thus, we choose two state-of-the-art studies – i) Saha et al. [102], and ii) Calefato et al. [20] related to our study. From the study of Saha et al. [102], we consider eight out of twelve features and discard the rest. We discard the four features such as – number of views, question score, number of favorite counts, and number of comments because they are not available at the submission time of a question. From the study of Calefato et al. [20], we consider all nine features. Next, we build four machine learning models using their attributes for each study. To create a level playing field, we also tune the parameters of baseline models to ensure the best performances. We finally compare the performance between baseline models and the proposed models. In particular, we compare the precision and overall accuracy between baseline models and our models.

Fig. 3.8 shows the comparative results between the proposed models and the baseline models. The performance of the models by Calefato et al. [20] is found comparatively poor than the others. Their accuracy and precision were found in the range of 50–52% for all the four algorithms. The strength of the features used by Saha et al. [102] is relatively higher than the features of Calefato et al. We see that the models trained with the features by Saha et al. outperform than that of the models by Calefato et al. The performance of the models by Saha et al. is about 1–7% higher than the models by Calefato et al. The overall accuracy of the models by Saha et al. is found 53–55% and the precision is found 53–59%. However, the proposed machine learning models outperform both the baseline models. The precision of the proposed models is about 6–25% higher than the models by Saha et al., and about 9–27% higher than the models by Calefato et al. Proposed models also have the improvement of overall accuracy between 5–25% from Saha et al., and 7–27% from Calefato et al.

We further attempt to investigate why the baseline models fail that much in predicting unanswered questions. We manually investigate 50 cases where both the baseline models fail. Our analysis shows that predictions often go wrong due to the length of the questions. Both the baseline studies consider the character length of the questions (title + body + code segment). Their models often detect the questions having long length as answered. However, our analysis suggests that questions with short texts and long code segments are often remain unanswered. In our study, we use the text-code ratio that might not suffer as the length of the questions do. The presence of code also misleads the baseline models. The models are trained in a way that the questions with code segment are more likely to be answered. However, we find several incorrect cases where questions do not have any code segments, and yet they get answered. We also find that the attributes such as the number of tags, presence of external link, submission time have almost identical values for both the question classes. Thus the baseline models find difficulty while classifying questions based on these attributes.

Generalizability of the Models' Performance. In our study, we attempt to extract such features that are not dependent on any programming languages. Here, we test the models to confirm the generalizability of their performance and the strength of the features. We select questions from two different programming

Table 3.4: Model Performance with C++ and Go

Technique	**Question**	**Precision**		**Accuracy**	
		C++	**Go**	**C++**	**Go**
Decision Trees	***Unanswered***	66.02%	65.28%	67.40%	66.0%
	Answered	69.03%	66.74%		
Random Forest	***Unanswered***	64.10%	62.38%	66.3%	64.02%
	Answered	69.31%	66.17%		

languages that are not used to train our models. In particular, we collect 100K questions (50K unanswered + 50K answered) of C++ programming language and 8K questions (4K unanswered + 4K answered) of Go programming languages. Go is an emerging programming language and thus we could not collect more questions. We select Random Forest and Decision Trees to test with C++ and Go since the experimental results (Table 3.3) show that they are well suited for our dataset and features.

The precision and accuracy of the models for C++ and Go reported at Table 3.4 show that the models perform reasonably well, although they lose some precision and accuracy (less than 15%). Our further investigation reveals a few causes that might explain why the performance has declined. We see that the code segments included in the questions related to the Go language are comparatively small than that of the Java and C# languages. On the contrary, the length of C++ code segments are fairly long but self-explanatory. The users often add a brief description. Moreover, Go is an emerging language, and most of the questions discuss theory and do not include any code segments. These scenarios above might affect the attributes such as text-code ratio, text readability. Despite having such discrepancies, the proposed model can predict the unanswered question with a maximum of about 66% precision and 67% accuracy. Such findings confirm the general acceptance of our features and models in predicting the unanswered questions.

3.4 Key Findings & Guidelines

Our study provides several insights that explain why a question might fail to receive answers at SO. These insights may guide the users of SO, especially, novice users to improve their questions and increase the chance of getting answers.

(F1) Question format matters. Proper formatting of the description in a question is essential. In particular, the different elements (e.g., text, code, hyperlink) are recommended to place inside the appropriate HTML tags to improve their visibility and readability. Otherwise, the question might be hard to read, which could force the expert users to leave the questions without answering.

(F2) Attach code segment when required. If a question describes a code related issue, it should include an example code segment to support the problem statement. We find that users often request for code segments in the comment thread while trying to answer the code related questions. More importantly, one should include non-redundant code that is able to reproduce the reported issues in the question description. According to an earlier finding, in SO, about 22% of the code segments fail to reproduce the reported issues in their questions [75]. In our manual analysis, we found that such lack of reproducibility could also leave a question unanswered.

(F3) Improper and non-technical description hurts. A clear description might prompt the expert users to answer a question. According to our investigation, we also find that the usage of appropriate technical terms is important. An ambiguous and non-technical description of a technical problem might mislead the users and thus hurt its chance of getting the solutions.

(F4) SO is not suitable for all topics. Several topics are more likely to be unanswered in SO. According to our analysis, top unanswered topics are primarily related to programming IDEs, libraries, and frameworks. A lot of unanswered questions from these topics are related to internal configurations and algorithms. To answer such questions, the creator and the maintaining team need to be active users at SO which they are not. Hence, many unanswered questions are redirected to Github issues through comments [82] where the members from the development team are involved. In addition to that, questions related to the hardware configuration of network devices are also more likely to be unanswered. Such questions can be resolved by IT experts who are not that much active at SO. During manual analysis, we see that 8% of unanswered questions are directed to other sites. Thus, the users should (1) avoid submitting the off-topic questions and (2) carefully select the tags for their questions.

(F5) Multiple issues in a single question hurts. Some users often discuss multiple issues in a single question, which is not recommended by the community members. According to our investigation, question with multiple issues or concerns often fail to receive the answers.

(F6) Unanswered questions are not always really unanswered. We find that several questions received working solutions in their comment threads. According to our manual investigation, 5% unanswered questions receive answers in the comment threads. Thus, although many questions are seen as unanswered, they are actually answered.

(F7) Miscellaneous findings. We also find several other causes that might leave the questions unanswered. First, several questions remain unanswered since they discuss outdated technology. Second, many unanswered questions were found with misleading titles. Third, users sometimes requested for suggestions instead of discussing problems clearly and asking particular solutions. Forth, some questions were perceived as duplicates to other questions.

3.5 Related Work

Several studies [5, 93, 94, 102] focus on classifying the SO questions. Saha et al. [102] introduce machine learning models to classify unanswered and answered questions of SO. Their models depend on the attributes (e.g., view count and scores) that are not available during the submission of questions. Thus, their models might fail to predict the unanswered questions during their submission. On the contrary, we investigate the question characteristics that are available during the submission of a question and construct prediction models to predict the unanswered questions. Saha et al. [102] also perform an initial investigation to understand why the questions of SO remain unanswered. They report that the lack of user interest is a major factor that might leave a question unanswered. We find that the user's interest often depends on the topics of the questions. The answering rate of several topics is fairly high, whereas several topics suffer from a low answering rate. We perform a comparative study on the question characteristics between answered and unanswered questions and report the top 10 topics (i.e., tags) which are more likely to remain unanswered.

Asaduzzaman et al. [5] conduct qualitative analysis to understand why questions of SO remain unanswered. They report several characteristics (e.g., impatient users) which are difficult to measure in practice. Besides, they do not provide any classification or prediction models for separating unanswered questions from the answered ones. Thus the strength of their features in classifying questions is not well understood. We automatically compute eight important features (e.g., topic response ratio) of SO questions and use them to predict the unanswered question. Furthermore, our prediction model outperforms two baseline models.

Calefato et al. [20] study a set of factors that are likely to influence the chance of getting answers by the questions at SO. They analyze about 87K questions and suggest that four factors might affect the chance of obtaining successful answers to the SO questions. They are presentation quality (e.g., presence of code snippets, question length), sentiment polarity (e.g., positive, negative), question posting time (e.g., day of week), and user reputation. According to our investigation, user reputation is one of the important attributes to predict unanswered questions. However, our feature set is more powerful for predicting the unanswered questions during their submission than those of Calefato et al. [20].

Ponzanelli et al. [93, 94] present an approach to classify the questions according to their quality (i.e., score). Our investigation reports that about 99% of questions having negative scores receive one or more answers. On the other hand, about 11% of questions which have positive score do not receive any answer [87]. Therefore, the classification model based on the score might not perform well in predicting the unanswered questions. Although a few of the features (e.g., metric entropy) are common between their model and ours, we extract more appropriate features to predict the unanswered questions.

Rahman and Roy [97] propose a model for predicting unresolved questions (i.e., no submitted answers were accepted as solutions by the questioner) of SO. Their feature analysis gives a key insight for understanding the difference between unresolved and resolved questions. We use a few of their features (e.g., topic entropy) in detecting unanswered questions. However, the remaining features (e.g., answer rejection ratio) are only effective in detecting the unresolved questions rather than the unanswered questions.

Chua and Banerjee [23] develop a conceptual framework known as the Quest-for-Answer to explain why some questions in Q&A sites draw answers while others remain ignored. They attempt to validate their framework empirically using 3000 questions (1500 unanswered + 1500 answered). However, some of their features are difficult to measure in practice (e.g., user politeness). A few of them (e.g., view count, question age) are also not available during the submission of questions. The remaining features (e.g., title length, description length) were employed by Calefato et al. [20]. However, our models outperform the models of Calefato et al. Wang et al. [135] investigate the factors for fast answers in technical Q&A sites. We not only investigate the causes of unanswered questions but also deliver machine learning models that can predict unanswered questions during their submission. To the best of our knowledge, no existing models are available to predict the unanswered questions during question submission time, which makes our work novel.

3.6 Threats to validity

This section discusses threats to the validity of this study. First, the baseline and the proposed models are built based on the four machine learning algorithms. One can argue that the results can vary with other algorithms. Saha et al. [102] use the Naive-Bayes algorithm in their study. However, the algorithm does not perform well in their study, and thus we discard this algorithm from our experiment. Furthermore, the algorithms used in our study are widely used by a number of related studies.

Second, we use five different quantitative metrics to measure text readability. Other text readability metrics can have different impacts on the results of our proposed models. However, we also analyze the result with other readability metrics and find that our reported metrics perform better than them.

Third, during the manual investigation of answered and unanswered questions in RQ2 (Section 3.3.2), the participants remain aware of the status of the questions. One can argue that, if the analysis were performed keeping the participants blind, the bias could have been avoided. However, we also make one of the participants blind about the status of the questions and perform manual analysis on a sample set. We do not find any significant inconsistency between blind and non-blind observations. Thus, such a threat might have been mitigated.

Finally, to balance the answered and unanswered questions in the dataset, we select random samples from the answered questions and all from the unanswered questions. Statistical conclusion validity might arise due to under-sampling the data [64]. However, we perform statistical tests routinely to justify our conclusions and thus avoid such threats.

3.7 Conclusion

A significant part of the questions of SO do not receive any answers. Predicting them beforehand can help one improve the questions and thus receive the answers in time. In this study, we contrast between unanswered and answered questions of SO quantitatively and qualitatively. We analyze 4.8 million questions and build models to predict the unanswered questions during their submission. Our contribution in this study is threefold. First, we conduct a quantitative analysis and find that topics discussed in a question, the experience of the question submitter, and readability of question texts are likely to predict whether a question will be answered or not. Second, we manually investigate 400 questions (200 unanswered + 200 answered) and demonstrate that our qualitative findings support the quantitative findings. Our manual analysis also reveals several non-trivial insights (e.g., avoid multiple issues in a single question) that could guide novice users to improve their questions. Third, we develop four machine learning models to predict the unanswered questions. Our models can predict the unanswered questions during their submission time with about 78% precision and about 79% overall accuracy, which are significantly higher than that of two baseline models. The models also perform reasonably well when we test them with questions from two different programming languages (i.e., Go, C++). Such performance also suggests the robustness of our models and features.

4 Can Issues Reported at SO Questions be Reproduced? An Exploratory Study

The first study in Chapter 3 investigates the unanswered and answered questions both manually and programmatically. In particular, compare unanswered questions with answered questions and attempt to find questions of SO remain unanswered. We find that topics discussed in the question, the experience of the question submitter, and readability of question texts could often determine whether a question would be answered or not. However, the irreproducibility of issues reported at the questions might prevent the questions from getting answers or appropriate questions. It should be noted that in the first study, we did not focus much on the reproducibility of the question issues that also may explain why questions of So remain unanswered or unresolved. In this chapter, we thus report an exploratory study on the reproducibility of the issues discussed in 400 questions of Stack Overflow. We also investigate the correlation between issue reproducibility status (of questions) and corresponding answer meta-data such as the presence of an accepted answer.

The rest of the chapter is organized as follows. In Section 4.2, we discuss two examples that motivate our idea of issue reproducibility. Section 4.3 discusses our study methodology. In Section 4.4, we classify reproducibility status of issues at SO questions, explore the challenges behind issue reproducibility, and then investigate their correlation with answer meta-data. Several key findings are discussed in Section 4.5. Section 4.6 focuses on the threats to validity, Section 4.7 discusses the related work and finally, Section 4.8 concludes the study.

4.1 Introduction

SO has become the ultimate platform for developers [93]. The number of users and the questions posted at SO are increasing exponentially [32]. A large number of questions contain such code segments that could potentially have issues (e.g., error, unexpected behavior) [127]. Once submitted, users at SO generally attempt to *reproduce* the programming issues discussed in the questions, and then submit their solutions. Reproducibility means a complete agreement between the reported issues and the investigated issues [137]. Unfortunately, such programming issues could not be always reproduced [76, 107] by other users which might prevent these questions from getting appropriate and prompt answers. Such scenario might also explain the 13.36% unanswered and 47.04% unresolved questions at SO [5, 32, 97]. Given these major challenges in question answering, a detailed investigation is warranted on the *reproducibility* of the code level issues that are reported at SO questions. Several existing studies [46, 147] investigate the usability and executability of the code segments posted on Q&A sites. Yang et al. [147] extract 914,974 code segments from the accepted answers of SO and analyse their parsability and compilability. However, their analysis was completely automatic, and they did not address the challenges of issue reproducibility. Horton and Parnin [46] analyze the executability of the Python code segments found on GitHub Gist system. They identify several flaws (e.g., import error, syntax errors, indentation error) that prevent the execution of such code segments. However, a simple execution success of the code does not necessarily guarantee the reproduction of the issues discussed at SO questions. Thus, their approach also fails to address the reproducibility challenges that we are dealing with. In short, all challenges of reproducibility could not be resolved using only automated analysis. Thus, there is a marked lack of research that (1) carefully investigates the challenges of issue reproducibility (of SO questions) and (2) develops automated approaches to help overcome such challenges.

In this study, we report an exploratory study on the *reproducibility* of the discussed programming issues at 400 questions from SO. For each of these questions, we follow three logical and sequential steps. First, we attempt to understand the issue of the question by analyzing its code segment and the associated textual description. Second, we clone the code segment in our development environment (e.g., IDE), and attempt to reproduce the reported issue. Third, we record our findings on the reproducibility of the issues from each of the questions. In the case of failure, we investigate why the issues could not be reproduced. We spent a total of 200 man-hours for our analysis in this study. Our findings from this study are two-fold. First, we find that 68% issues reported at SO questions can be reproduced and only 32% of them can be reproduced using the verbatim code from SO. The remaining code segments warrant either minor or major modifications in order to reproduce the issues. Second, we investigate the relationship between the reproducibility status of the issues (from questions) and the answer meta-data (e.g., accepted answer). We found that a question with reproducible issues has more than three times higher chance of getting an acceptable answer than the question with irreproducible issues. Our in-depth investigation and findings not only establish *issue reproducibility* as a question quality paradigm but also encourage automated tool supports for improving the code segments submitted at SO. We answer three research questions and thus make three contributions in this study as follows:

(RQ_1) What are the challenges in reproducing the issues reported at SO questions? How can issue reproducibility be measured? We conduct extensive manual analysis and investigate what types of actions are needed to reproduce the issues reported at SO questions. We classify the reproducibility status into two major categories (*reproducible, irreproducible*) and also discuss why the reported issues could not be reproduced using the submitted code examples.

(RQ_2) What proportion of reported issues at SO questions can be reproduced successfully? We conduct a detailed statistical analysis to determine what percentage of the issues can be reproduced and what percentage cannot be reproduced even after performing major modifications to their corresponding code segments.

(RQ_3) Does reproducibility of issues reported at SO questions help them get high-quality responses including the acceptable answers? We conduct a detailed investigation and determine the correlation between issue reproducibility (of questions) and corresponding answer meta-data such as presence of an accepted answer, time delay between posting of a question and accepted answer, and the number of answers.

4.2 Motivating Examples

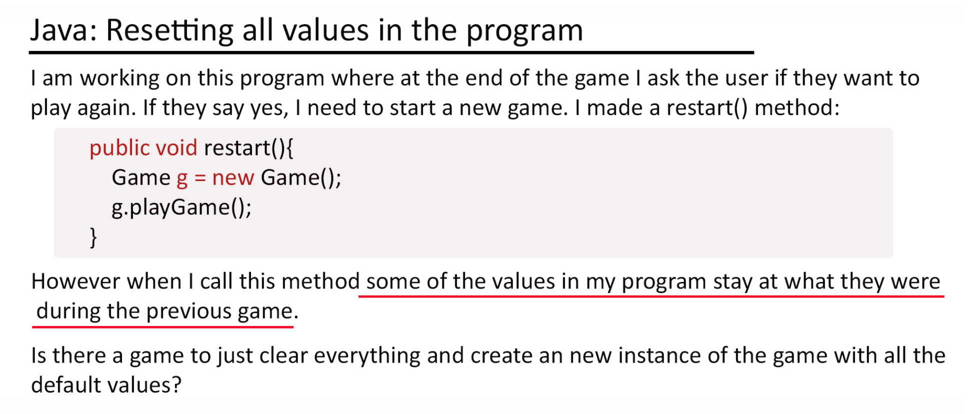
Code segments submitted as a part of the questions at SO might always not be sufficient enough to reproduce the reported issues. Let us consider the example question in Fig. 4.1. Here, the user attempts to reset all the variables while starting a new game. He discovers that the code is not working as expected and some of the variables are retaining their old values. Unfortunately, this issue cannot be reproduced since essential parts of the code are missing. For example, one user commented – *"Can you post more code? The Game class? The class that contains the restart() method?"* – while attempting to answer the question. In SO, this question has failed to receive a precise response. Even though the above code (i.e., Fig. 4.1) could be made parsable, compilable and executable with all necessary editing, the above reported issue could not be reproduced easily due to its complex nature. Thus, the automated analyses done by the earlier studies [46, 147] might also not be sufficient enough to overcome all the challenges of this reproducibility.

Let us consider another example question as shown in the Fig. 4.2. Suppose Alice, a software developer, wants to answer this question. First, she attempts to identify the issue and soon understands that the user is attempting to capture a person's full name (e.g., John Doe). Unfortunately, the user is getting only the first part of the given name (e.g., John). He invoked `next()` method of the `Scanner` class to take the input. In order to answer, Alice first copies the code segment to the IDE and then finds that the code does not even parse. As a result, the IDE returns several parsing and compilation errors. Fortunately, by performing several edits (e.g., addition of a demo class and main method) Alice could reproduce the stated issue. Similarly, this issue was also reproduced by the other users of SO, and the question received a high-quality answer within a couple of minutes. As the solution suggests, the user should have used `nextLine()` method instead of `next()` to avoid the reported issue.

4.3 Study Methodology

Fig. 5.3 shows the schematic diagram of our conducted exploratory study. We first randomly select 400 questions of SO, and then attempt to reproduce their discussed programming issues. In particular, we use

[1] https://stackoverflow.com/questions/798184
[2] https://stackoverflow.com/questions/19509647


Java: Resetting all values in the program

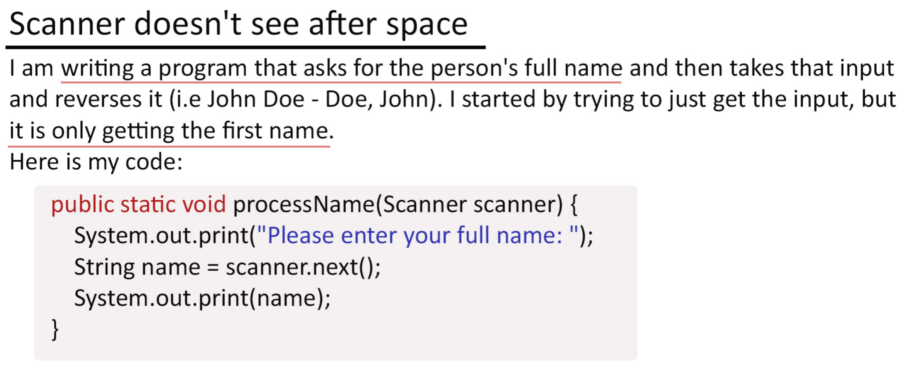
I am working on this program where at the end of the game I ask the user if they want to play again. If they say yes, I need to start a new game. I made a restart() method:

```
public void restart(){
   Game g = new Game();
   g.playGame();
}
```

However when I call this method some of the values in my program stay at what they were during the previous game.

Is there a game to just clear everything and create an new instance of the game with all the default values?

Figure 4.1: An example question[1] of SO. (**Reproducibility status:** *Irreproducible*)


Scanner doesn't see after space

I am writing a program that asks for the person's full name and then takes that input and reverses it (i.e John Doe - Doe, John). I started by trying to just get the input, but it is only getting the first name.
Here is my code:

```
public static void processName(Scanner scanner) {
   System.out.print("Please enter your full name: ");
   String name = scanner.next();
   System.out.print(name);
}
```

Figure 4.2: An example question[2] of SO. (**Reproducibility status:** *Reproducible*)

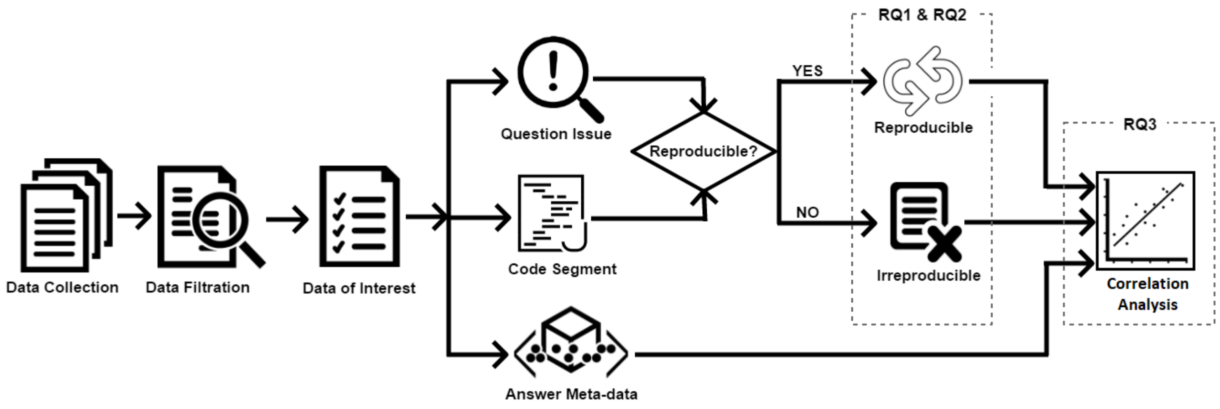


Figure 4.3: Schematic diagram of our exploratory study.

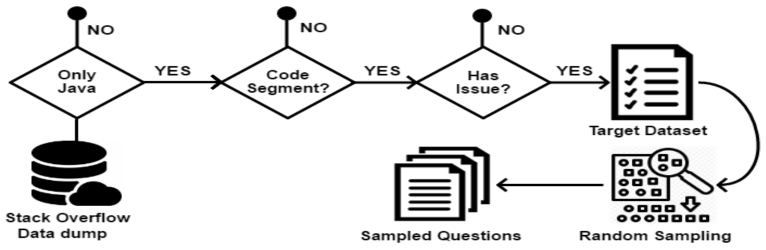


Figure 4.4: Selection of dataset for our study.

their submitted code segments, perform a list of edits, and then attempt to make them reproduce the reported issues. The following sections discuss different steps of our methodology.

4.3.1 Dataset Preparation

Fig. 4.4 depicts the data collection steps. We collect November 2018 data dump of SO from Stack Exchange site [32]. In particular, we select Java related questions since Java is one of the most popular and widely used programming languages. We collect a total of 85,876 questions that have only one tag namely <*java*> from the data dump. It should be noted that we impose this restriction on the question tags to (1) choose purely Java related questions and (2) keep our analysis simple. We then discard such questions that do not have any code segments. Since we deal with the reproducibility of the reported issues, the presence of code segments in the question is warranted. We thus consider only such questions that have at least one line of true Java code. According to our investigation, 70,103 out of 85,876 (i.e., 81.63%) questions have at least one line of code. We identify such questions using specialised HTML tags such as <*code*> under <*pre*>, and select them for our study.

Questions containing code segments might not always have an issue that needs to be reproduced during the answering time. Programmers might even seek general help or ask for more efficient source code than the submitted code segment. However, in this study, we target only such questions that discuss at least one issue each and contain at least code segment each. We thus carefully analyse the questions, and look for several keywords such as - *error, warning, issue, exception, fix, problem, wrong, fail* in the question texts. We then identify a total number of 30,528 questions that might have an issue. We randomly select 1,000 questions from them for manual analysis.

During manual analysis, we found a significant number of duplicate questions. In such cases, we consider the original questions with explicit issues and discard the duplicate questions. We also discard a few questions that were closed by SO due to their low quality. Overall, we manually investigate 400 Java related questions from 1000 random samples that meet our selection criteria, and answer our research questions.

4.3.2 Environment Setup

We use `Eclipse Oxygen.3a Release (4.7.3a)`[3] and `NetBeans 8.2`[4] to execute the code segments and reproduce the programming issues. Eclipse and NetBeans are two popular IDEs that are frequently used for Java programming. When the issues are related to compilation errors, we first employ `javac` to detect the compilation problems, and then use Eclipse and NetBeans to reproduce the issues. In particular, we attempt to find an exact match between our error messages and the ones mentioned in the questions of SO. We use `Java Development Kit(JDK)-1.8` as our development framework, `JavaParser`[5] as our custom AST parser, and MySQL Workbench[6] 8.0 as our example database for this study. We use a desktop computer having 64-bit Windows 10 Operating System (OS) and 16GB primary memory (i.e., RAM). We allocate 4GB as Java memory (Java heap for the IDE.).

4.3.3 Qualitative Analysis

Table 4.1
List of Code Editing Actions

- Exception handling	- Invocation of methods
- Code migration	- Debugging
- Addition of demo classes and methods	
- Inclusion of native and external libraries	
- Object creation, identifier declaration and initialisation	
- Deletion of redundant and erroneous statements	
- External file, database and dataset creation	

We, two of the authors, took part in the manual analysis and spent a total of 200 man-hours on the 400 questions. We follow a two-step approach for reproducing the issues reported at SO questions. First, we attempt to understand the reported issues clearly and identify the key problem statements from the question description. We also gather the supporting data such as input-output values, file format from the question texts. Second, using the code snippet and supporting data we attempt to reproduce the reported issues. We perform trivial, minor and major edits on the code segments in order to reproduce their issues. Table 4.1 shows our list of editing actions. We discuss each of the actions that are taken during qualitative analysis to make the code segments reproduce the reported issues as follows:

(a) *Addition of Demo Classes and Methods*: In SO, users often submit only the code examples of interest which are neither complete nor compilable. We wrap such code segments with a demo Java class and then

[3]https://www.eclipse.org
[4]https://netbeans.org
[5]http://javaparser.org
[6]https://www.mysql.com/products/workbench

place them under a main method. Main method acts as the entry point to the program.

In some cases, the code segment contains statements related to the creation of an object of a class or method invocation using an object while the definitions of the class and the method are absent. For instance, one of our code snippets has the following statements:

```
A obj_A = new A(x,y);
obj_A.add();
```

We see that the definition of the `class A` is absent. In order to make this code compilable, we define the class, add a constructor and also define the method add() within the class. Although the actual implementation is unknown, such modifications help us to resolve the compilation errors.

(b) *Inclusion of Native and External Libraries*: JDK comes with many libraries that help the developers accomplish many of the common tasks. However, developers frequently use external libraries for various specialized tasks. In SO questions, code segments that use classes and methods from native or external libraries often miss the import statements. We add the import statements associated with native libraries with the help of the IDE (e.g., Eclipse). In the case of external libraries, we look for relevant library references in the question texts. If such libraries were found, we include them in the IDE and then add necessary import statements.

(c) *Exception Handling*: Developers often submit question claiming that the code generates unexpected exception. We also find some questions whose issues are not related to Java exception but their code throws one or more exceptions. In that cases, we resolve them using appropriate exception handling.

(d) *Object Creation, Identifier Declaration and Initialization*: Undeclared identifier is one of the common compilation errors for the code segments submitted at SO. We declare the unresolved identifiers according to their types and initialize them with appropriate values according to the question specifications.

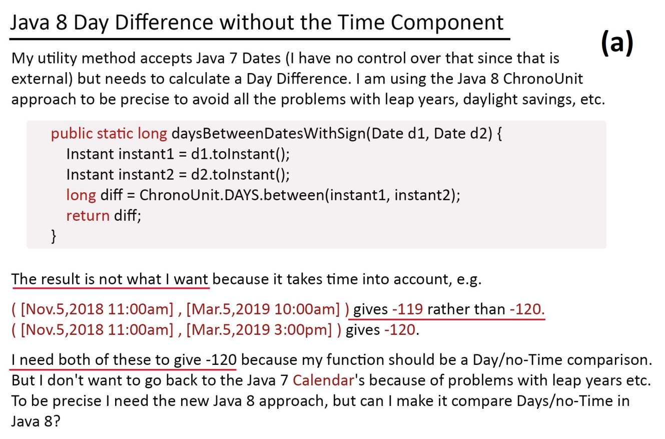
(e) *Invocation of Methods*: We identify several code snippets that have user defined methods but the methods were not invoked from anywhere. We thus add extra statements to call the methods if the issue reproduction warrants the execution of these methods. We also add appropriate parameters to call the methods.

(f) *Deletion of Redundant and Erroneous Statements*: We find several code segments that contain redundant and even erroneous code statements which are not related to the reported issues. We delete such statements or simply comment them out so that they are ignored by compilers and interpreters.

(g) *Code Migration*: We identify several code snippets that use outdated APIs which are not compatible with our environment. We replace the outdated APIs with the equivalent updated APIs. In some cases, we also invoke extra APIs to make the code compilable.

(h) *External File, Database and Dataset Creation*: We identify several code segments that accept input from `.txt` or `.csv` file. In such cases, we create one or more necessary files in our local drive and add the

[7]https://stackoverflow.com/questions/53159149



Java 8 Day Difference without the Time Component

(a)

My utility method accepts Java 7 Dates (I have no control over that since that is external) but needs to calculate a Day Difference. I am using the Java 8 ChronoUnit approach to be precise to avoid all the problems with leap years, daylight savings, etc.

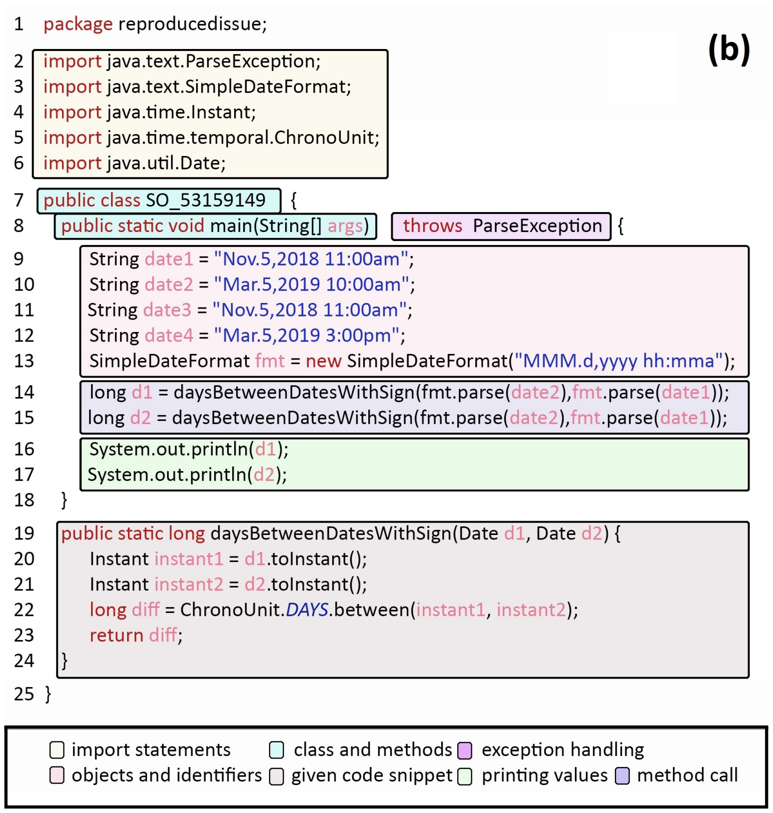
```
public static long daysBetweenDatesWithSign(Date d1, Date d2) {
   Instant instant1 = d1.toInstant();
   Instant instant2 = d2.toInstant();
   long diff = ChronoUnit.DAYS.between(instant1, instant2);
   return diff;
}
```

The result is not what I want because it takes time into account, e.g.

([Nov.5,2018 11:00am] , [Mar.5,2019 10:00am]) gives -119 rather than -120.
([Nov.5,2018 11:00am] , [Mar.5,2019 3:00pm]) gives -120.

I need both of these to give -120 because my function should be a Day/no-Time comparison. But I don't want to go back to the Java 7 Calendar's because of problems with leap years etc. To be precise I need the new Java 8 approach, but can I make it compare Days/no-Time in Java 8?

(a) An example question from SO with reproducible issue



(b)

```
package reproducedissue;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Date;
public class SO_53159149 {
  public static void main(String[] args) throws ParseException {
     String date1 = "Nov.5,2018 11:00am";
     String date2 = "Mar.5,2019 10:00am";
     String date3 = "Nov.5,2018 11:00am";
     String date4 = "Mar.5,2019 3:00pm";
     SimpleDateFormat fmt = new SimpleDateFormat("MMM.d,yyyy hh:mma");
     long d1 = daysBetweenDatesWithSign(fmt.parse(date2),fmt.parse(date1));
     long d2 = daysBetweenDatesWithSign(fmt.parse(date2),fmt.parse(date1));
     System.out.println(d1);
     System.out.println(d2);
  }
  public static long daysBetweenDatesWithSign(Date d1, Date d2) {
     Instant instant1 = d1.toInstant();
     Instant instant2 = d2.toInstant();
     long diff = ChronoUnit.DAYS.between(instant1, instant2);
     return diff;
  }
}
```

(b) The final code after necessary editing

Figure 4.5: An example question[7] from SO and the final source code after editing. (**Reproducibility status:** *Reproducible*)

sample data from the question so that we can verify the program correctness. Besides text file or spreadsheet, some programs require image file especially which are related to User Interface (UI). In such cases, we create images with specified format and dimension, and then execute the program. We also create a demo relational database (e.g., MySQL) and necessary tables when the code segments deal with database operations. Needless to say, we use the sample dataset provided by the question.

(i) *Debugging*: We identify several code segments that warrant for debugging to reproduce the issue. Sometimes programmers claim that they are getting unexpected behaviour from the code. For example, a developer reports that one of the `if` statements is never executed i.e., the expression of the `if` is always been false. Then we debug the code and check whether the if statement is really executed. When programmers claim that they are not getting expected values from an identifier, we usually print the value or debug the code to check the run time value of the identifier.

4.3.4 An Example of Issue Reproduction

Let us consider the question shown in the Fig. 4.5a. First, we attempt to identify the issue and soon understand that the user is trying to calculate the time delay (in days) between two given dates. Unfortunately, the user did not get the expected results for several input values. We then copy the code segment in the IDE and find that the code does not even parse. As a result, the IDE returns several parsing and compilation errors. Fortunately, by performing several actions we can reproduce the findings. First, we add a demo class (Fig. 4.5b, line No. 7) and place the method `daysBetweenDatesWithSignwith` within the class. Second, we add a main method (Fig. 4.5b, line No. 8). Third, we create four `String` objects (Fig. 4.5b, line No. 9-12) and initialize them with sample values according to the question description. Fourth, we then create an object `fmt` (Fig. 4.5b, line No. 13) of the class `SimpleDateFormat` to invoke the parse API that converts the text values stored in `date1, date2, date3` and `date4` to the type `Date`. Since `SimpleDateFormat` throws `ParseException`, we also handle the exception (Fig. 4.5b, latter part of line No. 8). Afterward, we invoke the given method and keep the return values in `d1` and `d2` of type `long` (Fig. 4.5b, line No. 14-15). Two inline variable declarations are also required here. We import the libraries for the classes `SimpleDateFormat, Date, ParseException, Instant` and also the enum `ChronoUnit` (Fig. 4.5b, line No. 2-6). We then execute the modified code, print the outputs and then check whether the outputs match with the ones reported by the user. Interestingly, the issue was reproducible.

4.4 Study Findings

We collect 400 questions from SO (Section 4.3.1), and analyse their code segments and textual descriptions. In particular, we attempt to reproduce the issues reported in these question texts by executing their corresponding code segments. We also ask three research questions in this study, and answer them carefully with the help of our empirical and qualitative findings as follows:

4.4.1 Answering RQ_1

RQ_1(a): What are the challenges in reproducing the issues reported at SO questions? We attempt to reproduce the reported issue discussed in the questions using the given code segment and other supporting information (e.g., data, instruction). However, we fail to reproduce them due to several non-trivial challenges. First, programmers often submit code segments that use methods from the classes which are not defined in the code segment. Despite adding appropriate class and method definitions, many issues cannot be reproduced. Second, code segments often miss such statements that are essential to reproduce the issues. For instance, reproducibility of an issue depends on the values of an array which are absent from the code segment. We could not reproduce such issues with the sample array values. Third, dependency on the external libraries is another major challenge towards issue reproducibility from the submitted code. In many cases, we do not find any hints that point to the appropriate libraries. We also identify such code segments that are too short to reproduce the issue. Too short code is also identified as the reason for getting no answer to SO questions [5]. We also identify several code segments that could not reproduce the issues due to their complex interactions with UI elements, databases and external files. We find a few segments containing outdated code (e.g., deprecated API) that cannot be replaced by alternative one. Several code segments are too long and noisy to reproduce the issues.

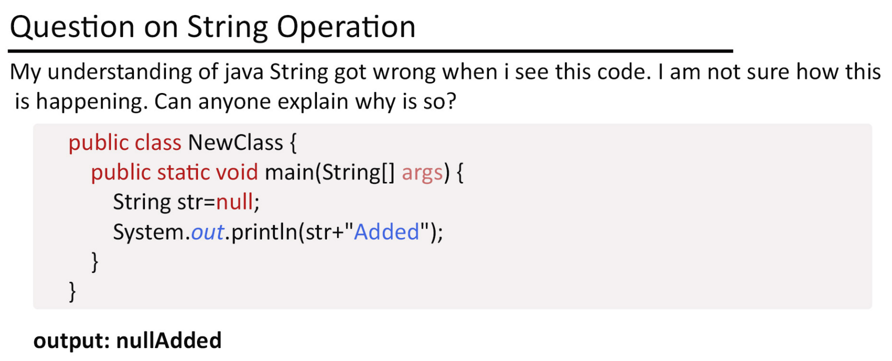
Table 4.2 shows the major challenges that prevent the reproduction of issues reported at SO questions. A question might experience multiple challenges. We note that half of irreproducible issues are plagued by undefined classes, interfaces and methods. That is, they are used in the code segments without proper definitions. We also note that about 40% issues could not be reproduced due to their missing import statements and missing external libraries.

Table 4.2
Challenges Preventing Issue Reproduction

Identified Reason	**Percentage**
Class/Interface/Method not found	51.00%
Important part of code missing	22.99%
External library not found	20.69%
Identifier/Object type not found	14.94%
Too short code snippet	12.64%
Miscellaneous	6.90%
Database/File/UI dependency	4.60%
Outdated code	1.15%

RQ_1(b): How can issue reproducibility be measured? We divide the issue reproducibility status

[8]https://stackoverflow.com/questions/1715533


Question on String Operation

My understanding of java String got wrong when i see this code. I am not sure how this is happening. Can anyone explain why is so?

```
public class NewClass {
  public static void main(String[] args) {
    String str=null;
    System.out.println(str+"Added");
  }
}
```

output: nullAdded

Figure 4.6: An example question from SO[8] (**Reproducibility status:** *reproducible without modification*)

(of questions) using two different dimensions. They are the success status of reproducibility and level of editing efforts in reproducing the issues reported at SO questions. Reproducibility status can be classified into *two* major categories: *reproducible* and *irreproducible*. In the following section, we discuss both categories in detail.

Reproducible (REP): Reproducibility of issues usually requires some modifications to the code segments. In some cases, issues can be reproduced from the given code segments without any modification. We therefore classify the *reproducible* status into *three* more sub-categories based on the complexity of the code level modifications, the level of human efforts spent and the time required to reproduce the issue. Fig. 4.7 shows our sub-classification of the *reproducible* category.

- *Reproducible without Modification (RWM):* The given code snippet is complete, stand-alone, and no explicit action is required to reproduce the issue. Fig. 4.6 shows a simple example of reproducible issue. Here, the user assigned *null* to a `String` object and then added to another string. Not surprisingly, he found that the *null* was also printed with the other string. To reproduce this issue, we just copy the code segment, paste it in the Eclipse IDE and then successfully reproduce the same output as noted in the question. Many of these issues are reported by the novice developers.

- *Reproducible with Minor Modification (RMM):* The issue can be reproduced from the code by performing one or more modifications that are comparatively less complex and less time consuming. Box A shows the low cost modifications added to the code. We spend about 15–30 minutes to reproduce each programming issue from this sub-category.

- *Reproducible with Major Modification (RMJM):* The reported issue can be reproduced from the code segment by performing one or more complex and time consuming modifications. Box B shows the high cost modifications. We spend about 30–60 minutes to reproduce each issue from this sub-category. Fig. 4.5 shows an example of reproducible issue where major modifications are performed on the code to reproduce the reported issue.

Irreproducible (IREP): These issues are less likely to reproduce even after several minor and major code level modifications. As mentioned above, two of the authors took part in the manual analysis. When one fails to reproduce the issue, the same question is analysed by the other author. If both fail, we then conclude the issue as irreproducible. In some cases, we spent even a few hours for a single question. Fig. 4.1 shows an example where the issue could not be reproduced since important details of the code/implementation are missing. Table 4.2 shows our identified challenges that prevent a programming related issue from reproducing.

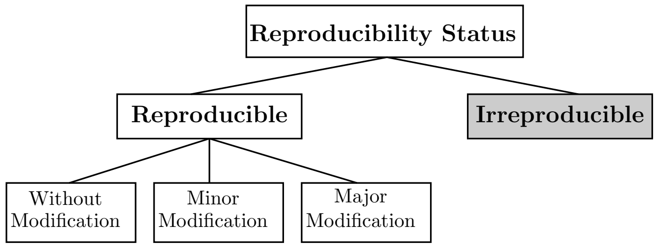


Figure 4.7: Classification of issue reproducibility status

Box A : Minor Modifications

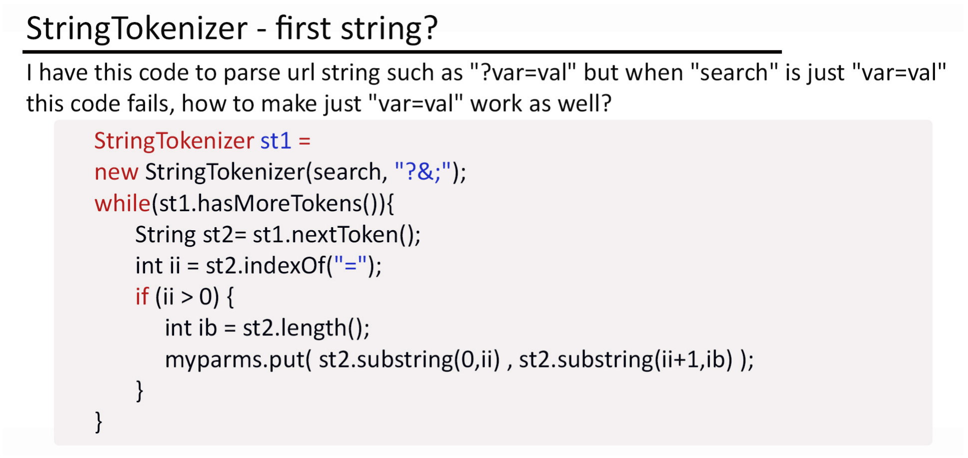
✛ Addition of a demo class or a main method or a method definition ✛ Creation of a constructor ✛ Declaration of an identifier/object whose type can be predicted easily. ✛ Initialization of an identifier/object by a default value or a sample value from the question ✛ Inclusion of the import statements (of native libraries) ✛ Handling an exception ✛ Resolve an external library dependency when the the code segment contains the import statements of them ✛ Resolve the less complex compilation errors ✛ Creation of the text/image/csv files ✛ Addition of sample values to the files from the question discussion ✛ Minor changes in the existing code ✛ Invoke a method with no parameters or known parameters.

Box B : Major Modifications

✛ Resolve external library dependency when the import statements are not in the code segment. ✛ Creation of database and table entries according to the problem description ✛ Major changes in existing code snippets ✛ Declaration of an identifier/object whose type cannot be predicted easily and their initialisation that demand code analysis ✛ Addition of sample input values to the files that are absent from question discussion ✛ Invoke a method with parameters where sample parameters are absent from the question description ✛ Code debugging

Besides the major classification above, we also consider the appropriateness of developers' claims about issues during the question submission. We found that there exist two more classes of issues that could not be reproduced as well. We discuss them in detail as follows:

Inaccurate Claim (IAC): In some cases, the stated programming issues in the SO questions might not be accurate. We call these issues Inaccurate Claim. The question shown in the Fig. 4.8 shows an example of inaccurate claim about programming issue. Here, the user raised an issue that he could parse the string


StringTokenizer - first string?

I have this code to parse url string such as "?var=val" but when "search" is just "var=val" this code fails, how to make just "var=val" work as well?

```
StringTokenizer st1 =
new StringTokenizer(search, "?&;");
while(st1.hasMoreTokens()){
    String st2= st1.nextToken();
    int ii = st2.indexOf("=");
    if (ii > 0) {
        int ib = st2.length();
        myparms.put( st2.substring(0,ii) , st2.substring(ii+1,ib) );
    }
}
```

Figure 4.8: An example question of SO[9] with inaccurate claim about the programming issue

like `?var=val` but could not parse the string like `var=val`. When we attempt to execute this code in our IDE, we find that both strings can be parsed successfully. Even two programmers also commented as follows: *'Sorry, that code doesn't fail'* and *'This works for me too'*. We find similar occurrences with run-time errors, compiler errors, unexpected behaviour reported by the users at SO questions. Such anomalies could have several explanations. First, this might happen due to the difference in the development environment. Second, users might fail to locate the defective code, and hence, they submit the wrong code fragments.

Ill-Defined Issue (IDEF): Ill-defined issues refer to the problems which cannot be reproduced consistently under any circumstances. That is, the question submitter does not specify the context precisely in which situation the alleged programming issue encounters. For instance, a user claimed that he was getting run-time exception after clicking buttons. During code execution we find several buttons in the user interface and some of them trigger runtime exceptions. However, since the user does not specify a button, the reported issue could also not be reproduced successfully.

4.4.2 Answering RQ_2

What proportion of reported issues at SO questions can be reproduced successfully? We classify the issue reproducibility status into two major categories which are again classified into further sub-categories. We investigate the reproducibility of programming issues discussed at 400 randomly sampled questions from SO. Fig. 4.9 shows the statistics on the reproducibility statuses of the questions. According to our analysis, 270 (67.50%) issues among 400 can be reproduced. On the contrary, 87 (21.75%) issues cannot be reproduced due to the challenges shown in Table 4.2. We find that 7.25% issues are claimed inaccurately. This might happen due to the lack of programming experience or proper analysis of the code during question submission. Unfortunately, we could not determine the reproducibility of issues for 14 (3.50%) questions. The users often fail to provide the appropriate contexts in their questions which are required to reproduce the reported issues.

[9]https://stackoverflow.com/questions/750557

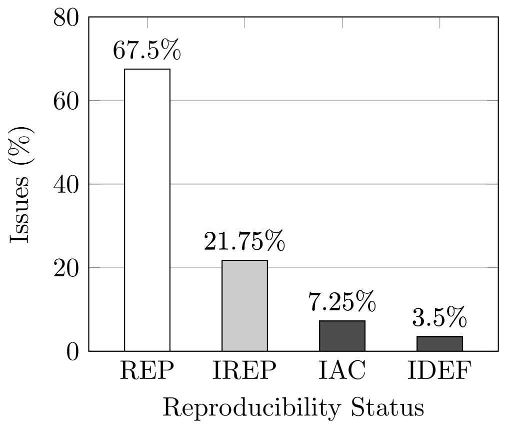


Figure 4.9: Issue reproducibility status. **REP** = Reproducible, **IREP** = Irreproducible, **IAC** = Inaccurate Claim, **IDEF** = Ill-Defined Issue

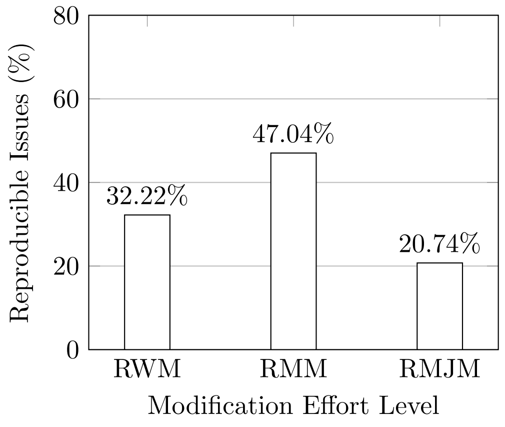


Figure 4.10: Reproducible issues and modification effort level. **RWM** = Reproducible Without Modification, **RMM** = Reproducible with Minor Modification, **RMJM** = Reproducible with Major Modification

We thus call them as ill-defined issues (IDEF).

The ratio of reproducible status according to the effort level is shown in Fig. 4.10. A half of them require minor modification whereas 20.74% issues require major modification to make the code snippets capable of reproducing the reported issues. Fortunately 32.22% snippets can reproduce the issues without any modifications.

4.4.3 Answering RQ$_3$

Reproduction of a programming issue discussed in the question texts allows one to experience the issue first hand. Such experience could help the users submit appropriate answers to the questions more promptly and more accurately. We thus analyze the relationship between reproducibility status of the reported issues at SO and the answer meta data such as presence of accepted answer, time delay between question and answer, and number of questions. In particular, we divide *RQ3* into three sub-questions, and answer them with detailed

Table 4.3
Issue Reproducibility Status vs. Presence of Accepted Answer (AA)

	AA Present	**AA Absent**	**Total**
Reproducible	201 (74.44%)	69 (25.56%)	270
Ireproducible	18 (20.69%)	69 (79.31%)	87
Total	219	138	357

Table 4.4
Issue Reproducibility Subcategory vs. Presence of Accepted Answer (AA)

	AA Present	**AA Absent**	**Total**
Without Modification	66 (75.86%)	21 (24.14%)	87
Minor Modification	102 (80.31%)	25 (19.69%)	127
Major Modification	33 (41.07%)	23 (58.93%)	56
Ireproducible	18 (20.69%)	69 (79.31%)	87
Total	219	138	357

statistics as follows:

RQ$_3$(a): Does reproducibility of issues discussed at SO questions encourage the acceptable answers? Table 4.3 shows the confusion matrix where the rows represent the reproducibility status (e.g., reproducible or irreproducible) and the columns represent the presence of accepted answer (e.g., present or absent). We note that 74.44% (201 out of 270) questions whose code segments are capable of reproducing the reported issues receive acceptable answers (i.e., solutions). On the contrary, only 20.69% (18 out of 87) questions with irreproducible issues receive the acceptable answers. Thus, the reproducibility of question issues increases the chance of getting a solution more than three times. We examine the correlation between the two categorical variables - reproducibility status and accepted answer presence. We use statistical test namely *Chi-Squared* test to measure the independence of these two categorical variables shown in Table 4.3. We find statistically significant *p-value* ($p - value = 0 < .05$). Thus, there is a strong positive correlation between the issue reproducibility of questions and their chance of getting the acceptable answers.

We also further analyse the reproducibility status based on effort level and determine their correlation with the accepted answer presence as shown in Table 4.4. We again find statistically significant *p-value* ($p - value = 0 < .05$) from the *Chi-Squared* test. Thus, the human efforts in reproducing the question issues are significantly correlated with the chance of getting the accepted answer. From the Table 4.4 we see that there is 75.86% (66 out of 87) chance of getting accepted answer when the issues can be directly reproduced from the verbatim code segment. On the contrary, the chance reduces to 58.93% (33 out of 56) when the submitted code needs major modifications. Surprisingly, the chance of getting accepted answer is highest (80.32%) for the questions whose code fragments require minor modifications to reproduce the issues. We thus further investigate this interesting scenario with more manual analysis. Three factors were identified that might explain such inconsistency:

Redundant and long code: The users at SO often submit questions that contain unnecessarily long code. Such questions often fail to get appropriate answers even though their code is capable to reproduce the reported issues without any modifications. According to our investigation, such code segments have more than 72 lines of code. We believe that such long code places extra burden on the other users during question answering. Thus, despite being reproducible without modifications, such code actually decreases the chance of getting an accepted answer.

Potential problem statements are not identified: Users often fail to mention the potential problematic statements within the submitted code in their questions. This increases the analysis time for the other users at SO while answering the question. Thus, despite the verbatim code segments being able to reproduce the question issues (without modifications), they might not receive the accepted answers.

Novice user: Sometimes novice users add a long code (e.g., an entire course assignment) without proper analysis. They even do not discuss about the issues properly. According to our analysis, 50% of users who added a long and redundant code and did not get an exact answer have a reputation score below 20.

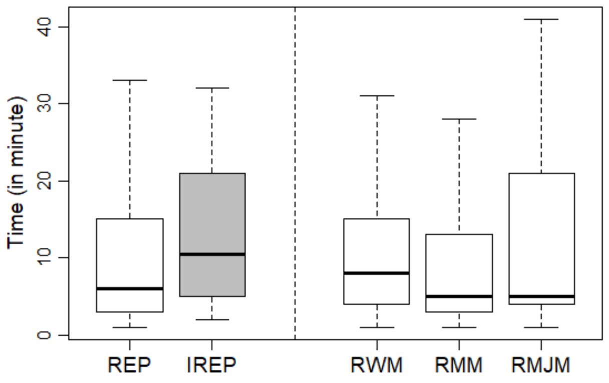


Figure 4.11: Time spent for reproducible and irreproducible issues

RQ$_3$(b): Does reproducibility of issues discussed at SO questions reduce the time delay of getting the accepted answers? According to RQ3a, there is a strong correlation between reproducibility status of question issues and the chance of getting an accepted answer. In fact, the chance is more than three times higher for the reproducible issues. In this section, we investigate whether the delay of getting the accepted answer could be influenced by the reproducibility status of the submitted issues at SO questions. We determine the delay between the submission time of a question and that of the accepted answer, and contrast between reproducible and irreproducible issues using such delays. Fig. 4.11 shows the box plots for the delay of getting the accepted answers. We see that the median delay of getting an accepted answer is about 5 minutes when the issue reported at the question is reproducible. On the contrary, such delay is almost double when the reported issue is not reproducible with the submitted code. We also find a significant difference in time delay for getting the accepted answer between reproducible and irreproducible status. We use *Mann-*

Whitney-Wilcoxon test, a non-parametric statistical significance test and get statistically significant *p-value* (p-value = .04 < .05). We also examine the effect size using *Cliff's delta* test and find large *effect size*, i.e., Cliff's |d| = 0.9365857 (large) with 95 percent confidence. Given all these evidences above, the delay of getting an acceptable answer is significantly higher for the questions with irreproducible issues. Thus, reproducibility of the reported issues is an important quality paradigm for SO questions, and such attribute could help them get the accepted answers quickly, even within 5 minutes.

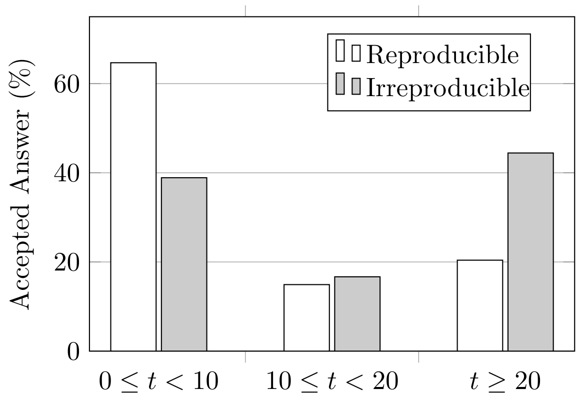


Figure 4.12: Accepted answer fraction vs. the time delay between question and accepted answer submission

Table 4.5
Reproducibility Status vs. Time Delay between Question and Accepted Answer Submission

	$t < 10$	$t \ge 10$	**Total**
Reproducible	130 (64.68%)	71 (35.32%)	201
Ireproducible	7 (38.89%)	11 (61.11%)	18
Total	137	82	219

Although the above box plots demonstrate the benefits of issue reproducibility, we further classify the delay of getting accepted answers into three intervals: $0 \le t < 10$, $10 \le t < 20$, $t \ge 20$. Fig. 4.12 shows the percentage of the accepted answers for reproducible and irreproducible issues against these intervals. We see that questions with reproducible issues receive about 65% of their accepted answers within only 10 minutes. Such percentage is only 39% for the questions with irreproducible issues. It also should be noted that 44% of these questions require more than 20 minutes on average to receive the accepted answers. Table 4.5 provides the raw statistics on the accepted answers and time interval. We examine the relation between these two categorical variables using statistical test namely *Chi-Squared* test. Thus, given all the evidences in above the tables and plots, issue reproducibility status is very likely to influence the time delay for getting the acceptable answers at SO.

RQ$_3$(c): Does reproducibility of issues discussed at SO questions encourage more answers? According to RQ3a and RQ3b, the reproducibility of reported issues at SO question might encourage quick

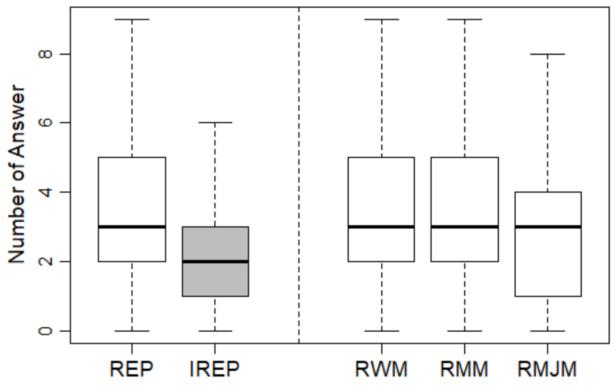


Figure 4.13: Number of answers for the questions with reproducible and irreproducible issues

and high quality responses from the users. In this section, we further investigate whether such reproducibility also encourages more answers at SO. Fig. 4.13 shows the box plots for the answer count of our selected questions against their issue reproducibility status. We clearly see that questions with issue reproducibility receive more answers on average than the counterpart. We also find a significant difference in the number of answers between two types of questions. *Mann-Whitney-Wilcoxon* test shows $p - value = 0$ whereas *Cliff's* $|d|$ *= 0.57 (large)* with 95% confidence. Given these statistical findings, we suggest that the reproducibility status of a reported issue at SO question has a significant impact on its chance of getting more answers.

4.5 Key Findings & Guidelines

While submitting a question at SO, it is recommended to add a bit of code fragment so that the reported program issues can be reproduced easily [110]. Experts users of SO also suggest to add complete and standalone code in the question [105]. However, our study delves further into the submitted code, and delivers more in-depth insights on the question issue reproducibility using such code.

(a) **Redundancy Hurts:** Only those statements should be added that are required to reproduce the reported issue and the redundant code should be avoided. Long and redundant code waste the developers time unnecessarily which might also hurt the question's chance of getting an accepted answer.

(b) **Dependency Matters:** Adding the import statements targeting the external libraries is very important. This is one of the major difficulties that we faced during the reproduction of question issues with the submitted code. The question texts also should point to the external libraries (if used) so that the users can include them in the IDE during issue reproduction.

(c) **Executable Code for Debugging:** The submitted code segment should compile and run if the reported issue requires debugging to reproduce. This is especially required when the program shows stochastic or unexpected behaviour. Without a reproducible code segment, such questions are generally hard to answer effectively.

(d) **One Issue Per Question:** Multiple issues should not be discussed using a single code segment.

Separate questions and code snippets are encouraged for separate programming issues for effective question answering.

4.6 Threats to Validity

Threats to *external validity* relate to the generalizablity of a technique. We manually analyse a limited number of questions and as such our results may not generalise to all the questions. However, we investigate a wide variety of questions of different types of issues in order to combat potential bias in our results. Nonetheless, replication of our study using additional questions and different languages may prove fruitful. Besides, we investigate only Java code segments. However, we believe that our insights can generalise to other statically-typed, compiled programming languages such as C++ and C#. But we caution readers to not over-generalise our results.

Threats to *internal validity* relate to experimental errors and biases [124]. The reproducibility status (e.g., reproducible, irreproducible) of the reported issue is threatened by the subjectivity of our classification approach. Thus, we cross-validate our results when an issue cannot be reproduced. We finalise the reproducibility status as irreproducible if we both fail to reproduce the question issue.

Threats to *construct validity* relate to suitability of evaluation metrics. We use *Mann-Whitney-Wilcoxon* test which is a widely used non-parametric test for evaluating the difference between two sample sets. However, the significance level might suffer due to the limited size of the samples. We thus consider the effect size along with the *p-value*. To see the correlation between two categorical variables we use *Chi-square* test. This statistical test of independence works well when there is a small number of categories (≤ 20) [72].

4.7 Related Work

There have been several studies on the reproducibility of software bugs [34, 76, 107, 137]. However, to the best of our knowledge, ours is the first work that investigates the reproducibility of the reported issues at SO questions using their submitted code.

Yang et al. [147] and Horton and Parnin [46] are the only closely related studies in terms of research methodologies and problem aspects. Yang et al. analyze the *usability* of 914,974 Java code snippets on SO and report that only 3.89% are parsable and 1.00% are compilable. They analyze the code segments found in only the accepted answers of SO and employ automated tool such as Eclipse JDT and ASTParser for the parsing and compiling the code. Then they report the errors that prevent the code segments from parsing and compiling without human involvement. Similarly, we analyze the Java code snippets from the questions of SO. However, unlike their approach which is completely automated, our approach is a combination of automatic and manual analysis. Not only we make the code parsable, compilable and runnable using appropriate modifications to the code, we also overcome the challenges to make them reproduce the reported issues at SO.

Horton and Parnin investigate the executability of Python code found on GitHub Gist system. Their primary focus was the execution of the Python snippets. However, we go beyond code execution and manually investigate the reproducibility of issues using Java code snippets submitted with SO questions. They also report the types of execution failures encountered while running Python gists. Similarly, we categorize the reproducibility status and identify the reasons why the issues could not be reproduced. Interestingly some reasons are common between ours and their study such as import error, syntax error. However, executability of a code does not always guarantee the reproducibility of an issue reported at SO question. Reproducibility requires testing and debugging which warrant for manual analysis, and this was not done by any of the earlier works. Besides, our research context differs from theirs since they examine gists shared on GitHub whereas we deal with code snippets found in SO questions.

Due to the growing popularity and importance of SO Q&A site, there have been several studies that focus on question/answer quality analysis. Duijn et al. [29] collect the Java code segments found in SO questions and suggest that several code level constructs (e.g., code length, keywords) are correlated to the quality of a code fragment. A number of studies investigate the quality of a code segment by measuring its readability [16, 17, 26, 95, 103**?**] and understandability [63, 104, 129]. Unfortunately, their capability of reproducing the issues reported at SO questions was not investigated by any of the early studies. This makes our work novel. As our empirical findings suggest, like readability and understandability, the reproducibility of issues could be considered as one of the major quality aspects of SO questions.

Several other studies [18, 20, 110, 135] investigate how to get a fast answer, create a high quality post or mine a successful answer. They suggest that information presentation, code-text ratio and question posting time are the key factors behind getting the high quality answers. Similarly, our findings show that reproducibility of an issue discussed in the question is likely to encourage high quality responses including the acceptable answers. Rahman and Roy [97] investigate why questions at SO remain unresolved. However, they also do not consider the issue reproducibility in their study.

4.8 Conclusion

In this study, we manually analyze 400 randomly selected questions from SO and investigate the reproducibility of their reported issues using their code segments. We answer three research questions in this work. We classify status of reproducibility into two major categories - reproducible and irreproducible. We find that 68% of issues can be reproduced using the submitted code after performing minor or major modifications to them. We also investigate and report why several issues could not be reproduced using the attached code segments. We then examine the correlation between the reproducibility of issues (of questions) and answer meta-data. Our findings suggest that reproducibility of question issues is likely to encourage more high quality responses including the acceptable answers. Thus, reproducibility of issues can also be treated as a novel metric of question quality for SO which was ignored by the earlier studies.

5 Rollback Edit Reasons and Ambiguities in SO

Our previous two studies (Chapter 3 and Chapter 4) (1) investigate the questions and propose early detection techniques of unanswered questions that outperforms the state-of-the-art techniques, and (2) expose the challenges of reproducibility and suggest that irreproducibility of question issues prevent them getting appropriate answers. Our second study also establishes issue reproducibility as a question quality paradigm. SO permits users to improve the quality of questions and answers by editing. In this study, we focus on the editing system of SO. In particular, we investigate why edits are rolled back and also the ambiguities of rollback edits.

The rest of the chapter is organized as follows. Section 5.2 discusses the overall study setup that comprises data collection, data preprocessing, data sampling and so on. Section 5.3 explores the rollback edits reasons, and Section 5.3.2 compares the rollback edits with the approved edits. Section 5.4.1 describes the ambiguities of rollback edits and the ambiguities detection algorithms are discussed in Section 5.4.2. Section 5.4.3 reports the ambiguities over the entire SO rollback ambiguities. Section 5.5 discusses the implications of our study, Section 5.6 discusses threats to validity. Section 5.7 focuses on the related works and finally Section 5.8 summarizes the chapter.

5.1 Introduction

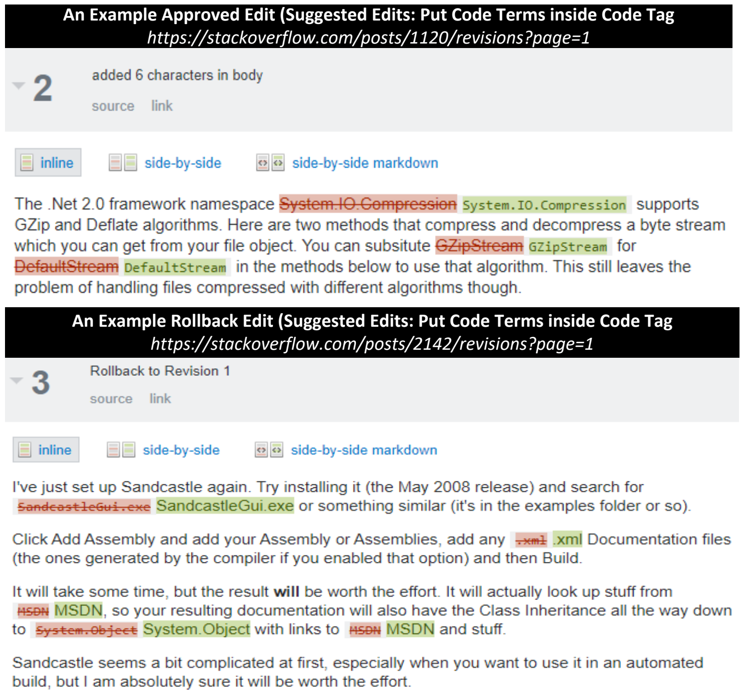
The adoption, growth, and continued success of a crowd-sourced forum, such as SO depend on two major factors, the participation of users and the quality of the shared knowledge [6, 56, 88]. Unlike traditional knowledge sharing venues and stakeholders (e.g., paid events), the developers participating in crowd-sourced forums are driven by the needs to learn from each other (which is often free), to become part of the community, as well as to feel the need to be recognized by peers in the community. Thus, the roles of knowledge seekers and providers in the crowd-sourced forums are often fluid and interchangeable. This elasticity leads to the design of a semi-decentralized system. However, the lack of authoritativeness can raise the concerns of trust on the quality of the shared contents [133]. The SO edit system is developed to give control to a content owner, while encouraging other users to suggest ways to improve the content. This *collaborative edit* approach is similar to other crowd-sourced platforms [25, 55, 58].

In SO, the edit system is introduced to promote quality, by allowing users to communicate on the quality of the shared knowledge (i.e., content) through editing of the content. In particular, collaborating editing helps to keep the questions and answers clear, relevant, and up-to-date. For example, users often edit questions and answers to fix grammar and spelling mistakes, clarify the meaning, and add related resources or hyperlinks. While any user can suggest an edit to shared content, SO adopts a semi-authoritative approach by allowing the content owner or privileged users (e.g., reputation $>$ 2K) to decide on the suggested edit. A suggested edit can be either approved (if satisfactory) or rejected (otherwise). In SO, rejected edits are called 'Rollback' edits.

The edits in SO were subject to a recent study by Wang et al. [134]. They identified 12 reasons why a suggested edit was rejected, such as incorrect code change, undesired code/text formatting, and so on. We find, though, reasons such as 'undesired formatting' could be prone to subjective bias. For example, consider the two suggested edits shown in Figure 5.1. Both edits suggest to put the code terms into a code block $< code >< /code >$. Both posts contained suggestions for Java API classes, `System.IO.Compression` in the first and `System.Object`. However, the first suggested edit is approved, while the second one was rejected. This observation leads to the question of how users are reviewing suggested edits and whether their judgment is clouded by individual bias. Lack of rules coupled with individual subjective bias could create *ambiguity* between the content owner (or privileged users) and the user who suggested the edit. Given that SO edit system relies on the voluntary (i.e., unpaid) cooperation from the users, such ambiguity may create confusion, frustrate the users, and dissuade the users from further contributing upon a rejection. It is thus necessary to understand the specific ambiguities that may be present in the rollback edits and use that to guide both forum designers and users.

We are not aware of any previous study that analyzed ambiguities in SO rollback edits. Clearly, such a study is warranted given recent focus on the quality of contents shared in SO [2, 48, 75, 97, 145, 151], the suite of tools and techniques developed to detect and recommend quality posts [20, 43, 59, 93, 94, 146], to produce software documentation [111] and to collect and summarize developers' reviews [130, 130, 131]. An understanding of ambiguities in rollback edits, for example, can guide research in software engineering to also analyze rollback edits for quality contents.

With a view to understand the ambiguities that may persist in the SO rollback edits, in this study, we first conduct a qualitative study of 1,532 randomly selected edits from SO, where 769 edits (385 question + 384 answer) were approved and 763 (382 question + 381 answer) were rejected. We analyze this dataset in two phases. In the first phase, we studied the reasons of rollback and approved edits. This phase produced a list of 14 rollback edit reasons and four major common actions based on which rollback and approved edits differ. Thus if any rollback edit showed suggestions similar to an approved edit, that could be due to an ambiguity in the suggested content or the inability of the edit system itself to guide user properly. In the second phase, we revisited the 763 rollback edits and exhaustively analyzed the edit history of each suggested prior edits to the post related to the edits. This analysis produced a catalog of eight ambiguity types, i.e., many of those edits were rejected due to one or more ambiguities. To understand the prevalence and evolution of those ambiguity types, we then developed eight algorithms, one each to detect the eight ambiguities. We ran the algorithms on the entire set of around 102K rollback edits we found in SO September 2019 data dump. In


An Example Approved Edit (Suggested Edits: Put Code Terms inside Code Tag
https://stackoverflow.com/posts/1120/revisions?page=1

2 added 6 characters in body

source link

inline side-by-side side-by-side markdown

The .Net 2.0 framework namespace ~~System.IO.Compression~~ `System.IO.Compression` supports GZip and Deflate algorithms. Here are two methods that compress and decompress a byte stream which you can get from your file object. You can subsitute ~~GZipStream~~ `GZipStream` for ~~DefaultStream~~ `DefaultStream` in the methods below to use that algorithm. This still leaves the problem of handling files compressed with different algorithms though.

An Example Rollback Edit (Suggested Edits: Put Code Terms inside Code Tag
https://stackoverflow.com/posts/2142/revisions?page=1

3 Rollback to Revision 1

source link

inline side-by-side side-by-side markdown

I've just set up Sandcastle again. Try installing it (the May 2008 release) and search for ~~`SandcastleGui.exe`~~ SandcastleGui.exe or something similar (it's in the examples folder or so).

Click Add Assembly and add your Assembly or Assemblies, add any ~~`.xml`~~ .xml Documentation files (the ones generated by the compiler if you enabled that option) and then Build.

It will take some time, but the result **will** be worth the effort. It will actually look up stuff from ~~`MSDN`~~ MSDN, so your resulting documentation will also have the Class Inheritance all the way down to ~~`System.Object`~~ System.Object with links to ~~`MSDN`~~ MSDN and stuff.

Sandcastle seems a bit complicated at first, especially when you want to use it in an automated build, but I am absolutely sure it will be worth the effort.

Figure 5.1: Similar suggestions in approved and rollback edits

Figure 5.2, we show the distribution of ambiguities in the rollback edits. In 2018, around 45% of edits rejected due to one or more ambiguity. This number was much smaller (only 34%) in 2009. Indeed, more and more edits have been rejected due to ambiguity since 2008. We observe that presentation ambiguity is the most prevalent which arises due to confusions among developers on how to present contents (e.g., code terms). We also see that temporal ambiguities are rapidly increasing in SO, which arise when multiple already approved edits to a post got rejected due to a newer suggested edits - which is then rejected, rendering the valuable contributions of the approved edits useless. We offer recommendations on how the edit mechanism in SO could be improved to promote better content quality.

5.2 Study Methodology

With a view to gain an understanding of the specific reasons contributing to rollback edits, we answer two research questions:

RQ1. Why do edits get rejected in SO?

RQ2. How do rollback edits differ from approved edits?

Our exploration of the rollback and approved edits produces factors to distinguish rollback and approved edits, and thus any rollback edits that deviated from those factors could have led to ambiguities. To understand the diversity and prevalence of ambiguities in rollback edits, we answered our last three research questions.

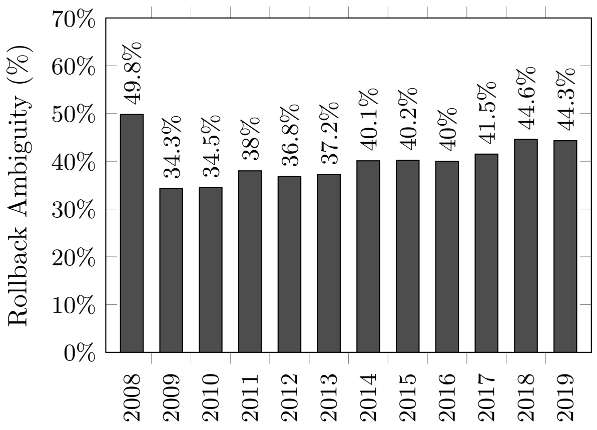


Figure 5.2: Distribution of rollback ambiguities over years

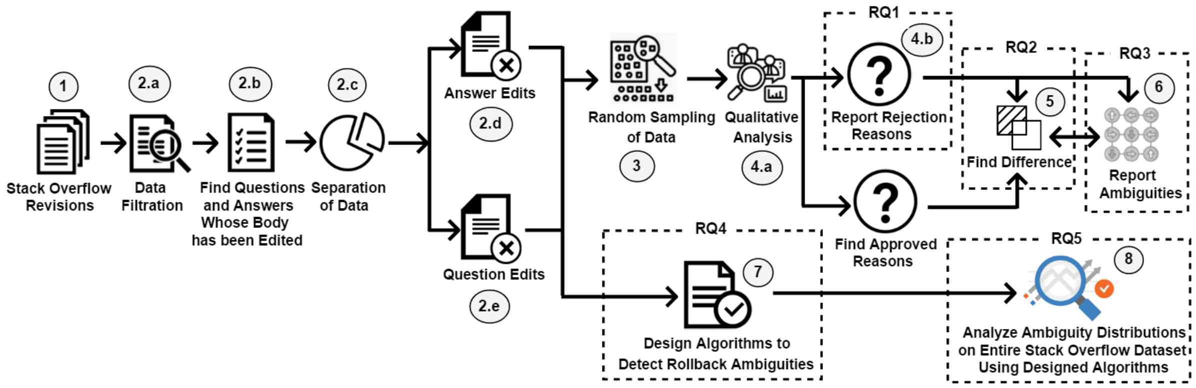


Figure 5.3: Schematic diagram of our conducted study.

RQ3. What are the ambiguities in the suggested edits that contribute to false/irrelevant edits?

RQ4. How can we automatically detect the ambiguities in SO rollback edits?

RQ5. How are the ambiguities distributed in the entire SO rollback edits?

Figure 5.3, we describe our overall methodology to answer the above five research questions. We describe the steps below.

• **Step 1. Data Collection** We download September 2019 data dump of SO from the Stack Exchange site [32]. The data dump contains all the essential information about questions and answers. It stores all the events (e.g., edit body, rollback body, post deleted) history of each question and answer. It also includes the date of each event, the user who triggered the event. The data dump only stores the revised question or the answer after each event. To detect the ambiguities programmatically, we require both the texts before and after each revision of questions or answers. We thus save the web pages of the SO site and parse them using HTML tags to extract the texts. In this study, we only investigate the revisions where users made body edits to the questions or answers. Our data dump contains 26,325,033 such revisions. Here, 26,208,560 edits (13,468,700 questions + 12739860 answers) were approved and 116473 edits (72,159 questions + 44,314 answers) were rejected.

• **Step 2. Data Preprocessing** There are 38 types of events that are tracked by SO [31]. In this study, we only investigate the revisions that are either connected to *Edit Body* or *Rollback Body* events that are performed on the body of a question or an answer (e.g., Fig. 5.3, Steps 2.a, 2.b). An *Edit Body* event indicates that the body of a question or an answer has changed. A *Rollback Body* event indicates that a question or an answer's body has been reverted to a previous version [134]. As we analyze the revisions (approved and rejected) of questions and answers individually, we separate the revisions according to their post type id (question =1 and answer =2) (e.g., Fig. 5.3, Steps 2.c, 2.d, 2.e).

All the revisions of a question or an answer along with the other related information (e.g., revision number, user, time, rollback to which revision) are found in a HTML table under the tag *<div id="revisions">*. To find our target data (e.g., revision texts, revision number, rollback to which revision), we first match the *Revision GUID* with the ids of the HTML table rows. Each revision has a unique GUID that we get from the data dump. Then we parse the appropriate HTML tags to get our target data. For example, we extract both the revision texts before and after rollback using the HTML tag *div class= post-text* under the tag *div class=sidebyside-diff.* We discard the records if any of the important data fields (e.g., revision number, user name) are missing or noisy. Finally, we get 63,078 (among 72,159) and 39,218 (among 44,314) revisions of questions and answers that were rejected respectively and are clean and complete. We analyze them programmatically.

• **Step 3. Random Sampling of Data for Qualitative Analysis.** To achieve a confidence level of 95% with a confidence interval of 5% [12], we randomly sampled (1) 385 approved question revisions from the entire approved question revision data set (e.g., 13,468,700 revisions), (2) 384 approved answer revisions from the entire approved answer revision data set (e.g., 12,739,860 revisions), (3) 382 rejected question revisions from 72,159 revisions, and (4) 381 rejected answer revisions from 44,314 revisions (e.g., Fig. 5.3, Step 3.a). To compute the size of our random sample, we use the following formula $\frac{Nz^2p(1-p)}{e^2N+z^2p(1-p)}$, where N is the population size (e.g., 13,468,700), z is the Z-score corresponding to a particular confidence level (e.g., 1.96 for a confidence level of 95%), e is the confidence interval (e.g., 5%), and p is population proportion (e.g., 0.5) [134].

• **Step 4. Identify Rejected Reasons.** The first two authors used open card sorting approach to label the reasons for each of 763 rollback edits [49]. In card sorting, the coders prepare cards (each denoting a label) for a given entity (i.e., an edit in our case) and discuss among themselves to decide on the most appropriate card for the entity by exhaustively analyzing the information related to the entity [49]. Axial coding is applied on the assigned labels to identify major categories. This step produced a list rollback edit reasons, which we report under **RQ1**.

• **Step 5. Distinguish Rollback Edits from Approved Edits.** We compare the rejected edits with approved edits to understand the common major actions in both of the edits. These observations offer us specific actions that are common in rollback and approved edits. The findings of this step are reported under **RQ2**.

• **Step 6. Identify Rollback Ambiguities.** With a deep understanding of rejected and approved edits, we revisit our dataset of rollback edits to understand whether any of those edits show specific commonality with approved edits or are recorded due to the inability of the platform to handle those edits. To determine the presence of ambiguities for a given rollback edit, we take into the following information: (1) The history of all suggested edits of the post prior to the rollback edit, and (2) Our list of common actions for both approved and rejected edit. Any rollback edit that did not follow the common actions, i.e., resembled an approved edit instead, were included into a bin of ambiguous edits. We then manually assigned a category to the ambiguity the rollback edit exhibited. We repeated the labeling process to form higher categories. The final list of ambiguity categories are reported under **RQ3**.

• **Step 7. Detect Ambiguities Automatically.** While our catalog of rollback ambiguity types offer insights based on 763 rollback edits, the total number of rollback edits in our SO data dump was around 102K. To analyze the prevalence of the ambiguities in all the rollback edits, we developed eight algorithms based on heuristics that we learned during our qualitative study (i.e., indicators of those ambiguities). We evaluate the algorithms and report their performance under **RQ4**.

• **Step 8. Study the Distribution of Ambiguities.** We apply our eight algorithms on the entire SO rollback edits. We compute the distribution of ambiguities in the edits and analyze their distribution over time. We report the findings under **RQ5**.

5.3 Exploring Rollback Edit Reasons

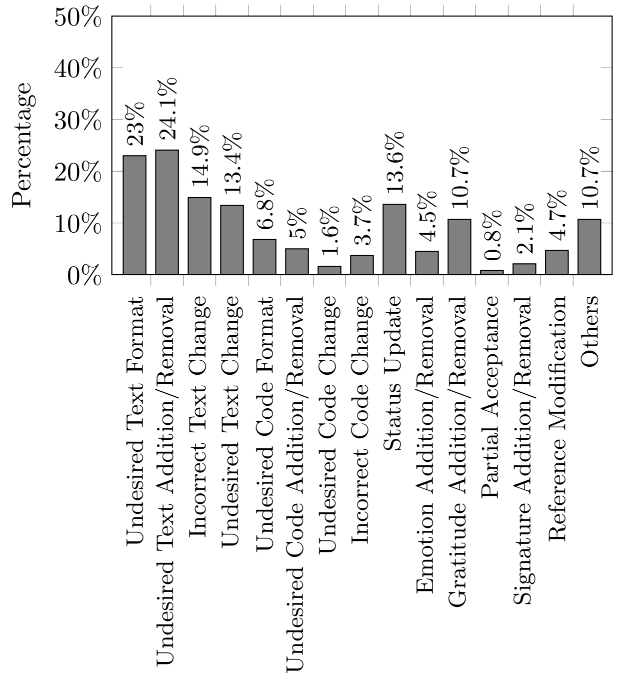
To properly analyze the potential ambiguities in rollback edits, we first need to learn the reasons contributing to edit rejections and approval in SO. We conducted a qualitative study of 763 rollback and 769 approved edits. In this section, we present the rollback edit reasons and then offer a qualitative comparison of major differences between rollback and approved edit reasons.

5.3.1 Rollback Edit Reasons (RQ1)

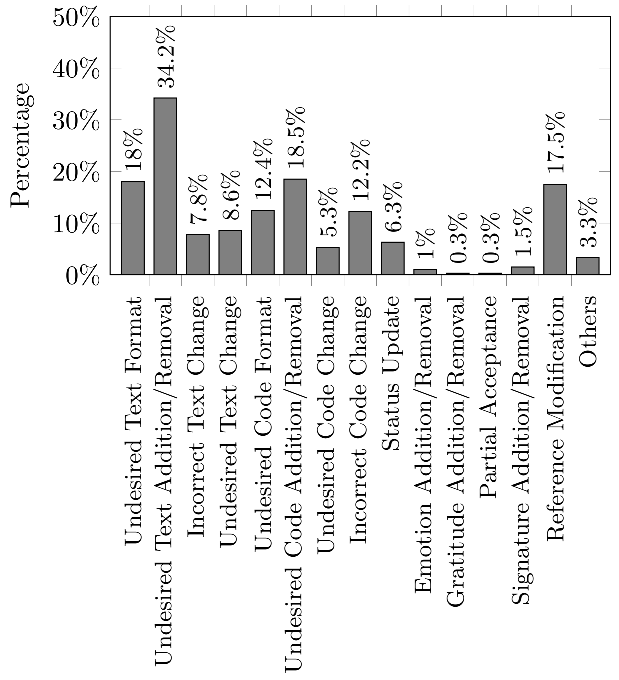
In Figure 5.4, we show the distribution of the rollback edit reasons in our manual analysis. We found all 12 rollback edit reasons previously observed by Wang et al. [134]. In addition, we found two new rollback edit reasons (Status Update, Gratitude add/remove). An edit can be rejected for multiple reasons, such as undesired text change and incorrect code change. In our dataset, about 41% answer edits were rolled back due to more than one reasons. Such percentage is about 27% for rollbacks of question edits.

(1) **Undesired Text Formatting.** Users often change the format of the texts unnecessarily. Such changes include changing the font, text cases, emphasizing text making them bold/italic, adding or removing space/newline, creating bullet/number list, and formatting text like code or vice versa. These kinds of undesired text formatting are often rolled back.

(2) **Undesired Text Addition/Removal.** Users add texts that have less or no impact on the quality of



(a) Question Rollback Reasons



(b) Answer Rollback Reasons

Figure 5.4: Summary of rollback reasons for questions and answers of our manually analyzed dataset.

questions or answers. That is, the additional texts do no contribute to improving the quality of questions or answers. For example, a user was added texts in support of a code segment as follows, *This code segment alone prints the Alphabet, no other action is needed as far as I am aware*[116]. However, such a decorative text is optional, thus the text is removed by a rollback. On the contrary, sometimes users remove important texts. Consider the following texts that carry essential messages on system configuration and operation. *Make sure* `php_curl.dll` *is in the directory listed under "extension_dir" in* `php.ini`*. If it is already, try restarting IIS (Apache always needs a restart from me when making php.ini changes) [113].* The texts were removed and then were brought back by a rollback.

(3) **Undesired Text Change.** Besides adding or removing texts, users sometimes make undesired changes in the existing texts. These changes include modification of the sentence structures (e.g., simple, complex), tenses (e.g., present, past), voices (e.g., active, passive). Rewording, interchanging contractions by root words, acronyms/abbreviations by elaborations and vice versa are also frequently seen text changes.

(4) **Incorrect Text Change.** Quality improvement is the primary goal of edits. Unfortunately, sometimes users make incorrect texts edits such as that might distort the meaning of the texts. Such edits include words with spelling mistakes, sentences with incorrect grammar, wrong wording and so on. For example [115], the sentence *"How do I get Mantissa and Exponent from double in jsp/java?"* has been changed to *"How do I get integer and fraction from double in jsp/java?"*. However, mantissa and exponent imply something different from integer and fraction. Incorrect changes also include grammatical and spelling mistakes, incorrect changes

of software versions or specifications, and incorrect changes of important terminologies. One user edits *F6* by *F5* [114], but the underlying task performed by pressing F6 is different from pressing F5. Thus, this incorrect edit rejected by a rollback.

(5) **Undesired Code Formatting.** Users make unwanted changes in code format, such as: (1) modification of code indentation, e.g., adding or removing spaces and newlines, (2) addition or removal of line numbers, (3) splitting or merging code segments. In addition to these, undesired formatting of code also includes changes in text cases. For example, one user was changed *"select"* to *"SELECT"* in a SQL query. However, the SQL query is case insensitive.

(6) **Undesired Code Addition/Removal.** In some cases, users add new code segments intending to clarify the discussed problem or to suggest an alternative solution for answers. They also add statements within existing code segments. When such codes neither enhance the quality of questions nor answers, they are rejected by rollbacks. On the contrary, users sometimes remove essential code statements that might make the code segments obsolete or change their functionalities. Removal of code segments from the questions might hurt the problem explanation.

(7) **Undesired Code Change.** Besides adding or removing codes, users also make undesired code changes. Such changes include refactoring (e.g., variable renaming), changing APIs, and editing comments. For example, a user substituted the statement – `int main(int argc, char* argv[])` by the statement – `int main(int argc, char *argv[])` [121]. Both of them are syntactically correct. Thus, such an unnecessary edit was rejected by a rollback.

(8) **Incorrect Code Change.** Sometimes users incorrectly change code. Such changes include changing types of variables, function return types, function arguments, arithmetic expressions. For example, a user substitutes a functions `int` return type by `void`. However, the function returns an integer value. Such an incorrect change was rejected by a rollback.

(9) **Status Update.** Users often update their status to clarify users confusion, attach important messages they missed to add during questions or answers submitting time, link external resources, and acknowledge users response. Consider the following status update where a user linked a resource of a Python client with their supported python versions. *Update: If you only need SOAP client, there is well maintained library called zeep* (https://python-zeep.readthedocs.io/en/latest)*. It supports both Python 2 and 3 [120].* Unfortunately, the above status was removed and thus it was brought back by a rollback. In some cases, unessential messages are attached to questions or answers by status updates. For example, the following status includes personal inquiry that might not become a part of a question. *Update: Could the people who are down-voting this question please post a comment or answer explaining what's so bad about it? If it's blindingly obvious, answer it and I'll vote you up [119].* Thus, the status update was rejected by a rollback.

(10) **Emotional Sentence Addition/Removal.** Users add emotional words/sentences that are often rejected by rollbacks. In some cases, they are brought back by rollbacks.

(11) **Gratitude Addition/Removal.** Sometimes users like to express their gratitude and appreciation at

the end of their questions or answers. According to our analysis, such thanksgiving words (e.g., thank you) are often rejected by rollbacks. Conversely, they are occasionally brought back by rollbacks.

(12) **Undesired Reference Modification.** Users add references such as hyperlinks, images or diagrams with their questions or answers. According to our investigation, the hyperlinks might inactive or points the wrong resources. The attached images might not an appropriate selection to clarify explanations. Thus, the references are removed by rollbacks. On the contrary, sometimes users either remove essential references or unreasonably modify them. Therefore, the edits are rejected by rollbacks.

(13) **Signature Addition/Removal.** Users add or remove signatures that include their name, ID, links of personal sites. In some cases, users only add their signatures at the bottom of questions or answers. That is, they do not make any noticeable changes in the body of the questions or answers except adding their names or personal links. We might consider such signatures as spam.

(14) **Partial Acceptance.** Partial acceptance refers to such a scenario when a revision is rolled back, but part of the changes are still accepted. Then, the accepted changes are included in later revisions.

(15) **Other.** Some other rejection reasons we observed were as follows.

(1) Add or remove duplication/deprecation notes. (2) Interchange the position of texts. (3) Introduce spam, i.e., users deface the post to promote product or service, insert garbage texts. (4) Ask questions in the answer. (5) Add solutions inside question.

5.3.2 Rollback Vs Approved Edits (RQ2)

Ambiguities may arise due to a lack of clarity while reviewing a suggested edit. With a view to establish a clear guideline based on empirical observations, we qualitatively study major differences between approved and rollback edit reasons. We use this guideline to analyze the sources of ambiguities in the rollback edits in Section 5.4. Our study shows four major actions for which approved and rollback edit contents differed from each other. Figure 5.5 shows the distribution of the actions between the rollback and approved edits in our study dataset. We summarize the actions below.

The most observed common action was 'Text Modification' in both edits (72% in rollback and 64% in approved). The modified texts in approved edits contained suggestions that are correct, offered concrete improvement over original text. In contrast, in rollback edits, such suggestions were mostly incorrect, and offered negligible or no improvement over the original text. The decision of how much of the suggested edits are significant for approval, relies on the content owner or privileged users. For example, the following edit was rejected due to undesired text modifications.

~~How to make class iterable?~~ Build a Basic Python Iterator

How would one create an ~~iterable~~iterative function ~~or class~~(or iterator object) in ~~Python~~python?

Actions related to 'Code modification' were found in 33% rollback and 18% approved edits. Similar to the 'Text modification', the approved edits contained correct code, *desired* code format, and *useful* code addition/removal. The following code segment was included in a question where the named keys of the array

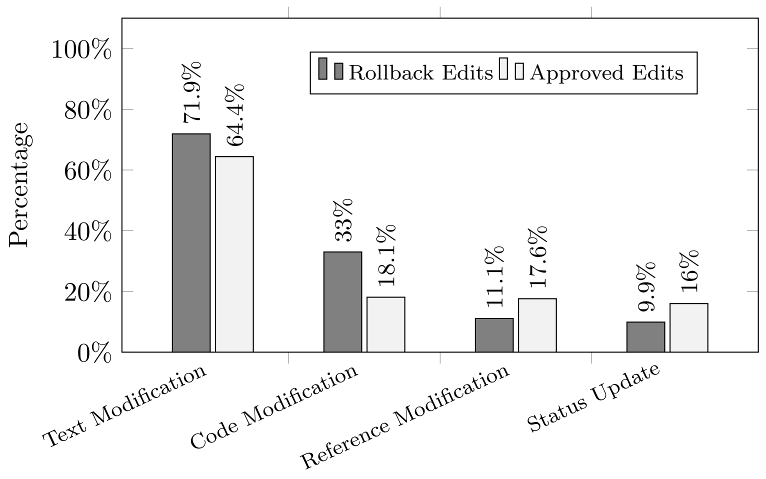


Figure 5.5: Comparison between rollback and approved edits.

`$item[]` were placed without quotes. The suggested edit was to place the keys (e.g., A,B) inside quotes. This was considered as a useful suggestion.

~~$items[A]~~$items['A'] = "Test";
~~$items[B]~~$items['B'] = "Test";
~~$items[C]~~$items['C'] = "Test";
~~$items[D]~~$items['D'] = "Test";

Actions related to 'Reference Modification' were covered more in approved than the rollback edits (18% vs 11%). In approved edits, such modifications could be mainly to correct an incorrect/inactive/obsolete link in the original content. In rollback edits, the suggested reference could be incorrect or were considered as *useless*. Consider the following example where reference of `SharpWired` was added. Unfortunately, the site could not be reached by the link. Such an inactive link was updated by edit and thus approved.

In our project, ~~SharpWired<http//sharpwired.sourceforge.net>~~ SharpWired<https://sharpwired.wordpress.com>, we're trying to create a download component similar to the download windows in Firefox or Safari.

About 10% of the status updates were rejected by rollbacks. On the contrary, about 15% were accepted. Approved edits related to 'Status Update' contained clarification messages in response to users' inquiry, informing others of achieving a self-solution, and acknowledging the positive contribution of others. In contrast, rollback edits with status update contained irrelevant inquiry, the addition of redundant information, removal of important status, messages in favor of the originality of questions.

Searching for Clarity Between Rollback and Approved Edit Reasons. As we noted above, the approval or rejection of a suggested edit can become subjective, such as when the content owner has to depend on his judgment to determine whether the suggestion is desired, significant enough, and useful. Given the diversity of programming languages, and the often opinionated and passionate nature of discussions that may be observed in software engineering forums [131], it is a non-trivial task for the SO forum designers or

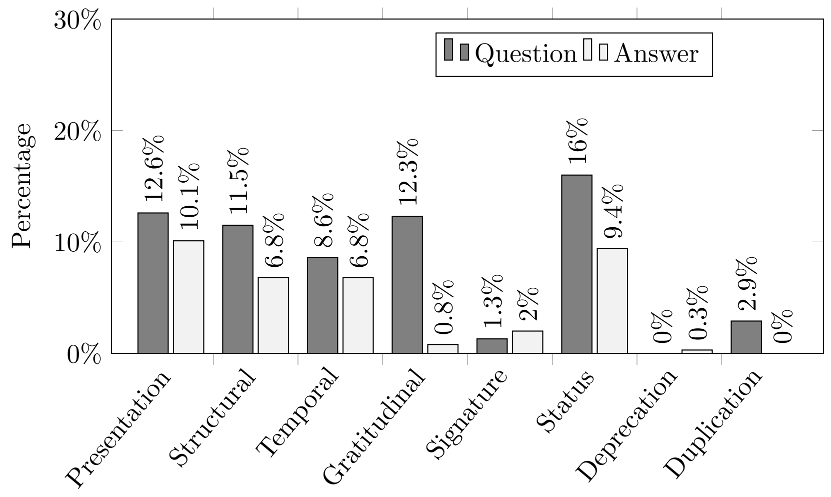


Figure 5.6: Summary of rollback edit ambiguities.

the user community in general to establish standard set of guidelines around the suggested edits. This lack of a commonly agreed guidelines can then create confusions among users while reviewing the edits. Therefore, ambiguities in the rollback edits may arise both from the observations of the same phenomenon (e.g., similar text edits) in both approved and rollback edits, as well as the inability of the SO platform to properly handle those reviews/rollback edits.

5.4 Rollback Edit Ambiguity

Our qualitative study in Section 5.3 hinted towards the presence of potential ambiguities in the SO rollback edits. With a view to understand the types of potential ambiguities, we first conduct a qualitative study of the rollback edits in our study dataset. Based on open coding of the differences between edited versions of a content before a rollback, we observed eight different types of ambiguities. We describe the ambiguities with examples in Section 5.4.1. Based on the insights gained, we then develop eight algorithms to detect the ambiguities automatically. We describe the algorithms in Section 5.4.2. We then apply the algorithms in entire list of around 102K rollback edits in SO. In Section 5.4.3, we discuss the prevalence, distribution, and evolution of the ambiguities across all SO rollback edits.

5.4.1 Rollback Ambiguity Types (RQ3)

In our qualitative study dataset of 763 rollback edits, we find that about 51% of rollback edits of questions have one or multiple ambiguities (about 31.1% in the answers). Figure 5.6 summarizes the ambiguities. A single rollback edit could exhibit multiple ambiguities: Among question rollback edits, 38.7% have single ambiguity and 12.3% have more than one ambiguity. For answers, the percentage is 26.6% for single ambiguity and 4.6% for multiple ambiguities. We explain the rollback ambiguities below.

(1) **Presentation Ambiguity.** Presentation ambiguity refers to the different presentation style of similar text or code elements in SO. For example, one user placed `myJSONObject` inside code tags. On the contrary, another user rejected such an edit and format `myJSONObject` as plain text. In some cases, users replace acronyms by its root words, then bring the acronyms back rejecting such edits. According to our manual analysis, we also see ambiguity for adding links. Sometimes links are added as regular texts, on the other hand, some users prefer hypertext. Some users make names (e.g., Java) and key terms (e.g., jQuery) of the questions or answers bold. However, users often reject such changes and then bring them back. We also see such ambiguity for changing text cases. For example, one user rejected .NET and replaced it by .net. Then such edit was rejected and brought .NET back. Even in a single revision, similar elements are formatted differently. However, for processing convenience, we only consider the presentation ambiguities of code elements. As shown in Figure 5.6, 12.6% rollback edits of questions have presentation ambiguity, whereas the percentage is 10.1% for answer rollback edits.

(2) **Structural Ambiguity.** A rollback reverts a post to a previous version in the edit history. Structural ambiguity refers to the rollbacks that revert a post to the immediate previous revision. Consider a rollback to revision 3 from 4. Revision 4 might be a revised (i.e., edited) version of revision 3 because there are no revisions to revert in between 3 and 4. Thus, such revisions might not be rollback revisions. In our manually analyzed dataset, we find 11.5% structural ambiguity for rollback edits of questions and 6.8% for rollback edits of answers.

(3) **Temporal Ambiguity.** Temporal ambiguity refers to such rollbacks when multiple approved edits are rejected by a single rollback. For example, a user rollbacks a revision from 12 to 7 [118]. That is, the revisions 8 through 11 were not that much needed. Figure 5.6 shows that temporal contributes 8.6% of total question rollback edits. Such percentage is 6.8% for answer rollbacks.

(4) **Gratitudinal Ambiguity.** Gratitudinal ambiguity refers to the dual viewpoint of accepting thanksgiving words. For example, a user added *thanks for your help* at the end of the question description [117]. This thankfulness was deleted by a rollback. Surprisingly, it was again brought back by a rollback. We see that 26.5% of total rollback edits have the gratitudinal ambiguity of question rollbacks, whereas such percentage is only 0.8% for answer rollbacks (Figure 5.6).

(5) **Signature Ambiguity.** Sometimes users add their names and affiliations after editing questions or answers. Such signatures often rejected by rollbacks. On the contrary, users sometimes bring such signatures back. As shown in Figure 5.6, signature ambiguities are found 1.3% for questions and 2% for answers among the total rollback edits in SO.

(6) **Status Ambiguity.** In the edit history, we see dual viewpoints accepting status updates by users. According to our manual analysis, status updates are occasionally rejected by rollbacks. On the contrary, rejected status updates are frequently seen to rollback by users. According to our manual analysis, status ambiguity was seen 16% of total rollback edits of questions and 9.4% for rollback edits of answers (Figure 5.6).

(7) **Deprecation Ambiguity.** This ambiguity discusses the ambivalent edits of deprecation notes. Addition

of deprecation notes within the body of questions or answers is often rejected by rollbacks. In some cases, they are also brought back by users. Deprecation ambiguity was seen infrequent in our dataset. Only 0.3% of total rollback edits were seen to have deprecation ambiguity of the rollback edits of answers. We did not see any such ambiguity in the rollback edits of questions.

(8) **Duplication Ambiguity.** This ambiguity discusses the ambivalent edits of duplication notes. Similar to deprecation, addition of duplication notes within the body of questions or answers is often rejected by rollbacks. In some cases, they are also brought back by some users. Although we did not see any duplication ambiguity in the rollback edits of answers, about 3% of total rollback edits of questions have duplication ambiguity.

5.4.2 Ambiguity Detection (RQ4)

We present algorithms to detect the rollback ambiguities we discuss in RQ3. We then evaluate the performance of the algorithms.

Algorithms. We developed eight algorithms, one each to detect the eight ambiguities.

(1) **Presentation Ambiguity.** SO users are suggested to add code elements (e.g., method name) inside the texts using <*code*>..</*code*> tags. Code segments are suggested to add inside <*code*> under <*pre*> tag. We see only the presentation ambiguity of code elements within the textual description, and thus remove the content under <pre> tag from the body of a question or an answer. Then we extract both the texts – before and after rollback using the <*div*> tag of class *post-text* (i.e., <div class = ”post-text”>). Next, we extract all of the code elements from both the texts and store them separately. We then examine whether the same elements are presented as code elements in one texts (e.g., before rollback) and non-code elements in another texts (e.g., after rollback) or not. That is, rollback either rejects presenting some elements as code elements (i.e., elements inside <code¿..</code> tag) or some code elements as non-code elements. We then count the rollback edits as ambiguous with presentation ambiguity.

(2) **Structural Ambiguity.** The information related to SO revisions of questions or answers is usually written in an HTML table. In particular, they can be extracted from a table row of class *revision* (e.g., <tr class=”revision”>). We first find the current revision number and the previous revision number where a rollback reverts in the edit history. Then we calculate the difference between the two revision numbers. We mark a rollback with structural ambiguity when the difference is one.

(3) **Temporal Ambiguity.** To detect temporal ambiguity, we find the current revision number and the previous revision number where a rollback reverts, and their difference, as discussed above. We consider a rollback is ambiguous with temporal ambiguity when the difference is more than two.

(4) **Gratitudinal Ambiguity.** We extract both the textual description of before and after rollback. We then look for several keywords related to gratitude such as – *welcome, thanks, sorry, appreciated, thank, ty* (i.e., thank you), *thx, regards,* and *tia* (i.e., thanks in advance) using regular expressions. When we find a keyword match to any of the above textual descriptions (not both), we consider the rollback as ambiguous rollback

Table 5.1: Performance of ambiguity detection algorithms

Ambiguity	Precision	Recall	F1-Score	Accuracy
Presentation	1	0.98	0.99	0.99
Structural	1	1	1	1
Temporal	1	1	1	1
Gratitudinal	1	1	1	1
Status	1	1	1	1
Duplication	1	0.94	0.97	0.99
Signature & Deprecation	0.33	1	0.5	0.99
Overall	0.99	0.99	0.99	0.99

with gratitudinal ambiguity. In particular, when rollbacks occur only to reject the gratitudinal phrases or bring such phrases back, we mark such rollbacks as ambiguous with gratitudinal ambiguity.

(5) **Signature Ambiguity.** To detect signature ambiguity, we first extract user information. In particular, we extract the names of two users: (1) rollback user, (2) user whose edit rejected by a rollback. We find these names using the <div> tag of class *user-details* (e.g., div class="user-details"). We then look for the names (full or part) in the textual part of both the edits – before and after rollbacks using regular expressions. When we find that any of the texts contain any of the two names, we consider such rollbacks as ambiguous with signature ambiguity. That is, when edits are rolled back due to add or remove names (i.e., signature), we mark the rollbacks have signature ambiguity.

(6) **Status Ambiguity.** Statuses are often updated followed by several keywords such as – *edit, update, note,* and *ps.* We look for these keywords in both the textual description – before and after rollback using regular expressions. When we find a match of any of the textual descriptions with a regular expression, we consider them as ambiguous rollbacks with status ambiguity. In particular, when rollbacks occur only to reject the statuses or bring them back, we mark such rollbacks as ambiguous with status ambiguity.

(7) **Deprecation Ambiguity.** To mark that an answer discusses deprecated technology (e.g., API), users add a message followed by keywords such as – *deprecation, deprecate.* First, we look for these keywords in both the textual description – before and after rollback using regular expressions. When we find a keyword match to any of the textual descriptions, we mark the rollbacks as ambiguous with deprecation ambiguity.

(8) **Duplication Ambiguity.** Similar to deprecation , to mark that a question of SO duplicates another, users add a duplicate message inside the body of the question followed by keywords such as – *duplicate, duplication.* To detect deprecation ambiguity, we first look for the keywords in both the textual description – before and after rollback using regular expressions. If we find a keyword match to any of the textual descriptions, we mark the rollbacks as ambiguous with duplication ambiguity.

Detection Accuracy. The high performance of our proposed algorithms is necessary to reliably analyze the distribution of the ambiguities in the entire SO dataset. We evaluated the performance of our proposed algorithms on an evaluation corpus. The corpus consists of 400 randomly sampled edits out of all the around 102K rollback edits in SO. The sample size is statistically significant with a 95% confidence level and 5 confidence interval. Out of the 400 edits, we pick 200 such edits where our algorithm did not find any ambiguity and 200 where our algorithm found one or more ambiguities.

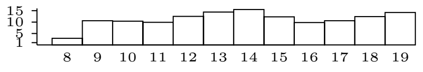
We manually label the sampled 400 rollback edits in a file as follows. 1. **Got**: the ambiguity detected by the algorithm. 2. **Expected**: the actual ambiguity based on our manual analysis. We then create a confusion matrix to analyze the performance of the algorithm as follows. 1. True Positive (TP) = 'got' ambiguity = 'expected' ambiguity 2. False Positive (FP) = ('got' ambiguity $\neq$ 'expected' ambiguity) or ('got' ambiguity but 'expected' no ambiguity) 3. True Negative (TN) = 'got' no ambiguity and 'expected' no ambiguity, and 4. False Negative (FN) = 'got' no ambiguity but 'expected' one or more ambiguity We use four standard metrics such as – Precision P, Recall R, F1-score $F1$, and Accuracy A [70] to compute the performance of each algorithm.

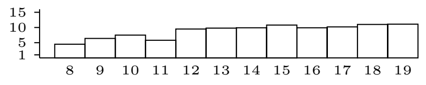
Table 5.1 shows the performance of our algorithms. Overall, our algorithms show high precision and recall (0.99). The precision is 1 for each algorithm except Signature & Deprecation. For the signature ambiguities, the misclassifications happened due to some users adding a different name (e.g., nickname) as a signature. That is, the name they added is different from their SO account name. Thus when we attempt to match the name with the body text of a question or answer, we did not find the name. However, when a rollback edit contains more than one ambiguities (e.g., temporal, status and deprecation [122]), our algorithms can successfully detect all of them.

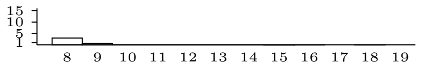
5.4.3 Ambiguity Distributions (RQ5)

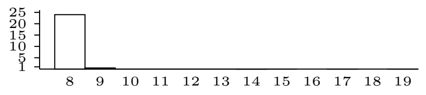
We applied our eight ambiguity detection algorithms on all of the 102,296 rollback edits in SO data dump of September 2019. In Figure 5.7, we show the distribution of the ambiguities in the dataset over the years between 2008 and 2019. SO was created in 2008. For each ambiguity, we show two distributions in Figure 5.7, one for rollback edits in questions and another for answers. Each bar chart in Figure 5.7 is interpreted as follows: The 'Percent' field informs that out of all the ambiguities detected in question and answer rollback edits, the presentation ambiguity in question rollback edits was observed 12.4% times. The 'Count' for Presentation ambiguity in Figure 5.7a tells that the the ambiguity was found 7791 times in rollback edits across the years. The bars in Figure 5.7a denote that out of the 7791 times the rollback edit was found, around 3% of those were in 2008 and around 15% in 2019.

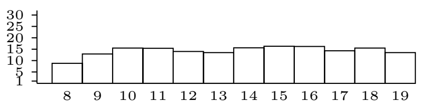
Presentation Ambiguities arise when similar code formatting were observed both in rollback and approved edits. Overall, the presentation ambiguities are much more in 2019 compared to the first year of SO (i.e., 2008). The number of question submissions and their edits were increased in 2019 compare to 2008. Interestingly, we see a decline in presentation ambiguity between 2009 and 2011, and then again in 2016. In the years 2010 and 2015, we find that experts of SO published a few articles on how to submit questions and

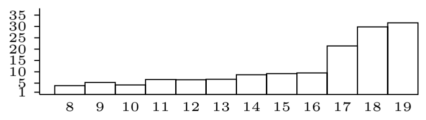

(a) Presentation (Questions) **[Count = 7791, Percent = 12.4%]**

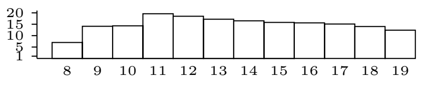

(b) Presentation (Answer) **[Count = 3884, Percent = 9.9%]**

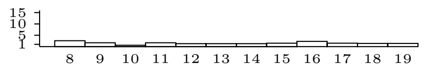

(c) Structural (Questions) **[Count = 61, Percent = 0.1%]**

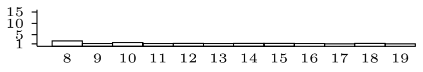

(d) Structural (Answer) **[Count = 44, Percent = 0.1%]**

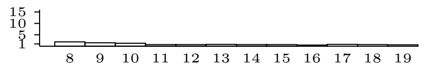

(e) Temporal (Questions) **[Count = 9434, Percent = 14.9%]**

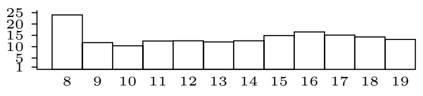

(f) Temporal (Answer) **[Count = 5914, Percent = 15.1%]**

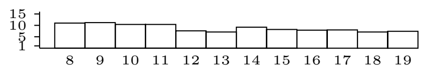

(g) Gratitudinal (Questions) **[Count = 9829, Percent = 15.6%]**

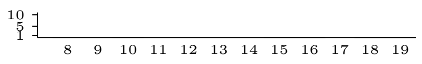

(h) Gratitudinal (Answer) **[Count = 590, Percent = 1.5%]**

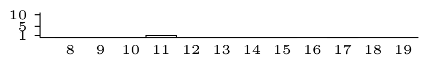

(i) Signature (Questions) **[Count = 756, Percent = 1.2%]**

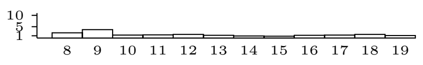

(j) Signature (Answer) **[Count = 301, Percent = 0.8%]**


(k) Status (Questions) **[Count = 8763, Percent = 13.9%]**


(l) Status (Answer) **[Count = 3173, Percent = 8.1%]**


(m) Deprecation (Questions) **[Count = 10, Percent = 0%]**


(n) Deprecation (Answer) **[Count = 3, Percent = 0%]**


(o) Duplication (Questions) **[Count = 782, Percent = 1.2%]**

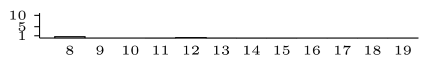

(p) Duplication (Answer) **[Count = 56, Percent = 0.1%]**

Figure 5.7: Distribution of Ambiguities in All Data

how the format different components (e.g., code snippet). After 2011, the number of presentation ambiguities started to rise again and it reached the maximum in 2014 (15.5%). We see increased adoption/discussion around new programming languages or their versions around this time. Clearly, each programming language has its own syntax.

The **Structural Ambiguities** (Figure 5.7b, 5.7c) were observed mainly in the formation years of SO (i.e., 2008, 2009), especially in the answers. After that they declined to almost zero. The ambiguities occurred in the first place due to users reverting to a previous version (i.e., N-1) instead of the current version (i.e., N) after rejecting an edit. Clearly, such ambiguities arose due to the unfamiliarity of the users with the SO platform.

The **Temporal Ambiguity** relates to the rejection of multiple previously approved revisions by a new rollback. We observe a sharp increase of temporal ambiguities in answer rollback edits in recent years (2018, 2019). A high percentage of temporal ambiguity suggests that users often perform less important or unwanted edits. Such edits do not improve the quality of questions or answers, and thus they could be rolled back together. As Wang et al. [134] observed, users performed more edits when they were closer to getting a badge. Thus users in SO are suggesting less useful answer edits recently compared to before.

Rollbacks edits with rejecting gratitudinal phrases or bringing them back are marked as ambiguous with **Gratitudinal Ambiguity**. Such ambiguities were seen frequent in question rollbacks but not much in answers (Figure 5.7g, 5.7h). It increased from 2008 to 2010, and then declined gradually. Both **Signature** and **Duplication** ambiguities were observed infrequently and in very low frequency. Similarly, **Deprecation ambiguity** was rarely seen. **Status ambiguity** was seen with about 14% for questions and 8% for answers on average. For questions, status ambiguity was very high (24%) in the year 2008.

5.5 Implications of Findings

The findings from our study can guide the following major stakeholders in crowd-sourced platforms: (1) **Forum Designers** to improve the edit system, (2) **Forum Users** to guide their edit behavior, and (3) **Researchers** to study collaborative editing.

• **Forum Designers.** Given that content quality is paramount to the success of SO, it is necessary for SO to enhance the edit system by addressing the shortcomings we found. Proper measures can reduce the dominant ambiguities, such as presentation, status, and temporal ambiguities. The temporal ambiguity, in particular, removes previously approved edits to a content, and thus can be detrimental to the overall quality of the shared content as well as to the motivation of the affected users. Such ambiguities also generate more suggested/rollback edits, while very little of those could be meaningful. From a site management perspective, this inefficiency can cost both the performance as well as overall reputation of SO.

• **Forum Users.** Our catalog of edit rejection reasons can be used to teach forum users. Similarly, the content owner or privileged users (i.e., the user who approves/rejects an edit) can learn from our rollback edit ambiguity catalogs and become more mindful before making a decision.

• **Researchers.** The content quality in SO has been the subject of a number of recent research papers [94, 151]. Studies traditionally look at the current version of a post to develop tools and techniques, such as assessment of API misuse patterns [151], or producing live API documentation [111]. Given the edit history of a post is also available and given that we see quality edits of a post can get rejected due to ambiguities, we recommend that research in software engineering also analyzes the suggested edits as well to produce potentially better and sound documentation and recommendations out of SO data.

5.6 Threats to Validity

• **External Validity** threats relate to the generalizability of our findings. In this study, we focus on SO, which is one of the largest and most popular Q&A websites for developers. Our findings may not generalize to other non-technical Q&A websites that do not focus on software development. To minimize the bias when conducting our qualitative analysis, we took statistically representative samples of all relevant revisions with a 95% confidence level and a 5% confidence interval [12].

• **Internal Validity** threats relate to experimenter bias and errors while conducting the analysis. Our study involved qualitative analyses of revisions in RQ1 and RQ3. To reduce the bias, each revision was labeled by two of the authors and discrepancies were discussed until a consensus was reached. We automatically detect the ambiguities on a large data set. The automatic detection results are mostly similar to the results of the manual investigation. This agreement also mitigates the bias of manual analysis. A few results varied because the sampled data may not always properly represent the entire data set.

• **Construct Validity** threats relate to the difficulty in finding data relevant to identify rollback edits and ambiguities. Hence, we use revisions of the body of questions and answers from the Stack Exchange data dump, which we think are reasonable and reliable for capturing the reasons and ambiguities of revisions. We also parse the web pages to create a large data set to apply our ambiguity detection algorithms. However, we discard the incomplete and noisy records to keep our data set clean and reliable.

5.7 Related Work

Related work can broadly be divided into the two areas: (1) Collaborative editing, and (2)Improving Content Quality in Q&A Sites.

5.7.1 Collaborative Editing in Q&A Sites

Collaborative editing systems are common in Wikipedia [55, 58], GitHub code editing [25], webcasts [78], scientific contents [22, 66], and so on. Studies show that collaborating editing positively impacts towards the improvement of shared contents [55, 58]. A recent study by Wang et al. [134] in SO found that users are motivated to edit more when they are closer to getting a badge. Indeed, offering incentives as reputation scores is found to be useful to improve post quality [58]. Similar finding was also observed webcasts by Munteanu et al. [78], who tested the effectiveness of engage users to collaborate in a wiki-like environment to edit/correct transcripts that are produced from webcasts through an automated speech recognition system. Kittur et al. [55] find that the increase in the number of editors does not guarantee the quality of the articles in Wikipedia.

Our study offers complementary viewpoints to the above studies, in particular to Wang et al. [134]. While we find rollback edit reasons similar to them, we also observed new reasons. Unlike the above work, we report with both qualitative and quantitative evidences, for the first time, on the diverse types of rollback edit ambiguities. Similar study can be carried out to find ambiguities in other collaborative editing sites.

5.7.2 Improving Content Quality in Q&A Sites

The success of crowd-sourced forums depends both on user participation and shared content quality [2]. Quite a few studies analyzed the quality of knowledge shared in crowd-sourced forums. The quality of question is important to get an answer: Lack of clarity, relatedness, and reproducibility of the problem, as well as the too short question could dissuade developers from answering to the question [5, 75]. The reputation and past activity of an asker could also factor into the likelihood of a question getting resolved [97]. As such factors of good questions are investigated, such as code to text ratio, etc. [20, 29]. However, depending on the platforms and user characteristics these factors can vary [48]. As such, it is important to detect content quality automatically and guide developers for a given platform. Ponzanelli et al. [93, 94] studied characteristics of low quality posts in SO and developed classifiers to detect those automatically. In contrast, high quality posts were automatically detected by Ya et al. [146] to assist in answering similar questions. During their study of revisions in SO, Wang et al. [134] found that users who make more edits in a short time, are likely to get more edits rejected. Thus while good edits can improve the content quality, bad edits can harm the content quality. In fact, both monetary of reputation-based incentives can attract both low and high quality responses [53, 134].

Our study findings emphasizes that the editing system, in particular, the rejection of edit process is suffering from diverse critical reasons and ambiguities. Without proper handling of those reasons, ambiguities will persist and may increase in the system. This could then harm the overall content quality.

5.8 Conclusions

• **Summary.** The edit system in SO encourages developers to improve the posted content. The owner of a content or privileged users who have moderation privilege can approve or reject a suggested edit. With a view to understand the reasons behind rejected edits, we qualitatively analyzed 1,532 edits in SO. We found 14 rollback reasons and eight types of ambiguities that are prevalent in the rejected edits. The ambiguities can confuse developers while edit suggestions. To determine the prevalence of the ambiguities, we developed algorithms to automatically detect that ambiguities that showed high degree of precision (average 0.99). We ran the algorithms on all rollback edits in SO. We found several ambiguities (e.g., presentation of contents) are consistently present in the edits, while ambiguities such as temporal are rapidly rising (i.e., accidental removal of already approved edits due to a more recent rejected edit).

• **Implications.** Our findings can guide 1. forum designers to properly develop edit system to promote and ease the sharing of quality contents, 2. forum users to understand the contributing and ambiguous factors towards rollback edits, and 3. researchers in software engineering to investigate tools and techniques to guide forum users and designers to handle the rollback edits properly.

6 Do Subjectivity and Objectivity Always Agree? A Case Study with SO Questions

To assess the quality of questions and answers (a.k.a., posts), SO Q&A site relies on collaborative voting. The quality of the questions and answers is subjectively evaluated by users through this voting mechanism. While analyzing the questions and answers of SO (in the previous studies), we find several counter-intuitive findings that cast serious doubts on the reliability of the evaluation mechanism employed at SO. In this study, we thus focus on the assessment quality rather than focusing on content quality. To ensure a non-biased, reliable quality assessment mechanism, we attempt to verify the subjective evaluation mechanism of a post with the corresponding objective evaluation.

The rest of the chapter is organized as follows. In section 6.2 we discuss the methodology of our study. In section 6.3 we discuss how we construct the datasets we use for our analysis. In section 6.4 we describe the metric selection ground. In section 6.5 we perform an in-depth comparative study between promoted and discouraged questions with respect to the results of our selected metrics. In section 6.6 we present our prediction models and the results we obtain. We discuss the overall findings in section 6.7. In Section 6.8 we survey related work. In section 6.9 we discuss the threats to validity, and finally draw our conclusions in section 6.10.

6.1 Introduction

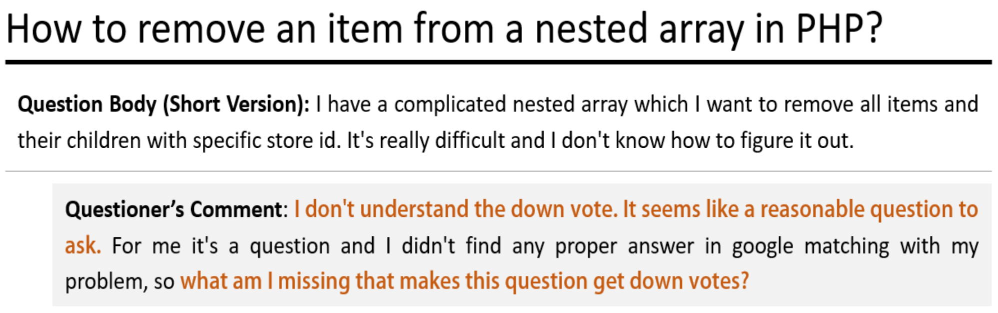
SO has emerged as one of the largest and the most popular Q&A sites for programming problems and solutions. It has been a valuable knowledge base containing about 16.7 million questions and 26 millions answers as of December 2018, which can be leveraged for various problem solving during software development [127]. The quality of any question or answer in SO is subjectively evaluated by the users through a voting mechanism. High-quality posts are generally voted up, whereas the vague, unclear or ambiguous posts are likely to be voted down. The net votes (upvotes − downvotes) cast against a given post in SO forms an evaluation metric called *score*, which also approximates the subjective quality of the post. However, according to our investigation, 48% of the questions that received working solutions have a score below zero [87]. That is, they receive more downvotes than upvotes. In other words, although these questions were valid and were answered successfully, they might not be considered of high quality by others. About 18% of the accepted answers (i.e., verified solutions) also fail to receive the maximum votes among all answers in a Q&A

thread [87]. That is, the best answers might not have been chosen as the solutions. The users of SO are also often disappointed by the evaluation against their questions (e.g., Fig. 1) and answers (e.g., Fig. 2). All these findings above suggest the highly subjective nature of the votes cast by the users in SO. Honsel et al. [45] also debunked several common myths (e.g., long answers get more downvotes) on the quality assessment system of SO. These myths were not substantiated by the empirical evidence. Furthermore, several previous studies also show that the incentive systems of Q&A sites may always not drive certain users in a positive way. For example, opportunistic users may find loopholes, and aggressively game the system for their advantage [47, 52, 134]. All these findings above cast serious doubts on the reliability of the evaluation mechanism employed at SO. In order to ensure a fair, reliable and effective evaluation mechanism, an investigation is warranted that verifies the subjective evaluation of a post (by the SO users) with the corresponding objective evaluation. Objective evaluation involves numerical metrics that estimate the quality by minimizing subjective bias. Such an agreement analysis between subjective evaluation and objective evaluation might help us better understand the potential or peril of the voting systems employed in SO.

Several studies [18, 30, 79, 126] analyse questions and answers of SO and attempt to assess their quality. Nasehi et al. [79] analyse the quality of code segments included in SO posts, and study the aspects that make up a good code example. Treude and Robillard [126] investigate to what extent developers perceive such code examples as self-explanatory. The appropriateness of code explanation is further studied by Ercan et al. [30] using Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [36]. Calefato et al. [18] focus on the four light weight, user-oriented features (e.g., user reputation) to predict the high-quality answers from technical Q&A sites. Novielli et al. [80] argue that the emotional style of a technical question could be a quality aspect that might influence the answering time of the question. Thus, the above studies simply rely on the light weight, ad-hoc attributes to estimate a post's quality. A few other studies [29, 93, 94] attempt to separate up-voted questions from down-voted questions using both the subjective (e.g., user reputation) and objective (e.g., text readability) metrics. While our work overlaps with them in terms of methodology, our research goals are different. In particular, we attempt to verify whether the subjective evaluation makes sense and the high-scored (hereby, promoted) questions on SO are actually preferable to the low-scored (hereby, discouraged) ones in terms of different objective quality metrics.

In this article, we conduct a case study with 2.5 million questions of SO where we verify the subjective quality of SO questions with objective evaluation. In particular, we check whether the quality of questions perceived by users matches the objective quality estimate. We have several findings from this agreement analysis. We found that four of the metrics totally agree, one metric somewhat agrees, and two metrics totally disagree with the subjective evaluation of the quality of a post. We also develop classification and prediction models based on the objective measures that outperforms the state-of-the-art classification models.

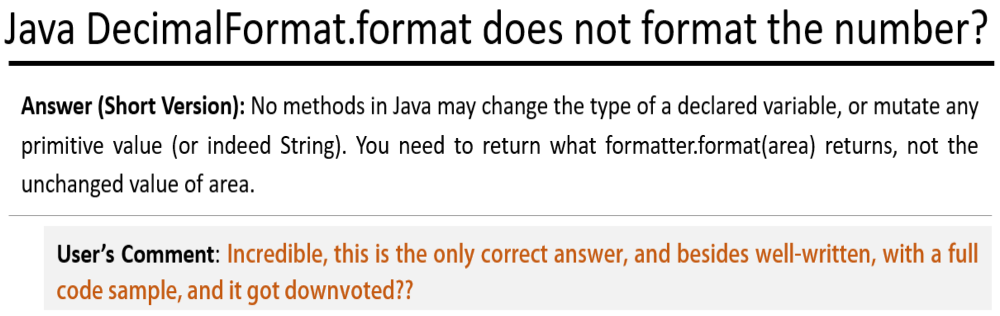
[1]https://stackoverflow.com/questions/47183288
[2]https://stackoverflow.com/questions/11765616


How to remove an item from a nested array in PHP?

Question Body (Short Version): I have a complicated nested array which I want to remove all items and their children with specific store id. It's really difficult and I don't know how to figure it out.

Questioner's Comment: I don't understand the down vote. It seems like a reasonable question to ask. For me it's a question and I didn't find any proper answer in google matching with my problem, so what am I missing that makes this question get down votes?

Figure 6.1: An example of a question submitter complaining in a comment that the subjective evaluation his question (hence, included the question[1] and the discussions in the comments) is not reasonable.


Java DecimalFormat.format does not format the number?

Answer (Short Version): No methods in Java may change the type of a declared variable, or mutate any primitive value (or indeed String). You need to return what formatter.format(area) returns, not the unchanged value of area.

User's Comment: Incredible, this is the only correct answer, and besides well-written, with a full code sample, and it got downvoted??

Figure 6.2: An example of a user complaining in a comment that the answer (hence, included the question[2], answer and the discussions in the comments) must not be voted down.

SO might incorporate a mechanism to estimate the objective quality of posts using our metrics. Then users could asses the quality of the posts at SO and also their post quality during submission time. Alternatively, a client-side browser plug-in can be developed employing our model. By installing the plug-in, users can estimate the objective quality not only their existing posts but also their new posts. In this article, we answer two research questions and thus make two contributions as follows:

(RQ_1) Does crowd-based subjective assessment of the quality of SO questions agree with the objective assessment of their quality? We conduct a comparative analysis between the promoted and discouraged questions using ten objective metrics from the literature. Our analysis reports several findings. First, topic entropy, metric entropy, text-code ratio, and text-code correlation support the subjective evaluation. That is, promoted questions are specific, less ambiguous, use unusual terms, and their code segments are well explained with accompanying prose. Second, the reusability of the code segment and text readability do not agree with their subjective evaluation. That is, the code examples included in promoted questions are hard to reuse. They are less parsable than those of the discouraged examples. Code examples included in discouraged questions are complete and more parsable. However, they often have redundant statements that might cause additional time in code analysis. According to our investigation, promoted questions often contain program elements (e.g., API classes) within the texts. The existing readability metrics consider such elements as complex words that hurt readability. Third, code understandability could

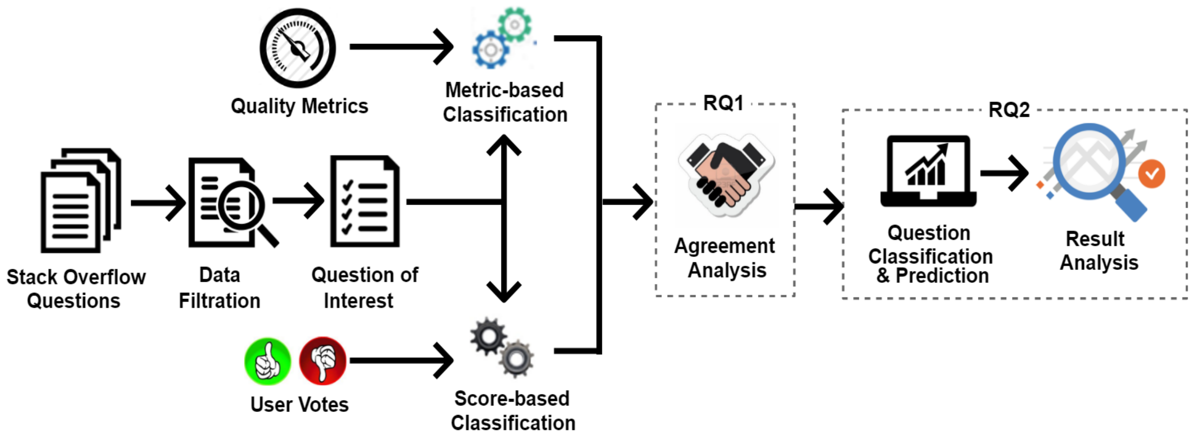


Figure 6.3: Schematic diagram of our study.

be either higher or lower for the promoted questions based on the programming languages. Forth, title quality, code readability and sentiment polarity are almost similar for both the promoted and discouraged questions.

(RQ_2) Can we classify the promoted and discouraged questions of SO Q&A site? Can we predict them while the users are posting the questions? We develop five machine learning models to classify the promoted and discouraged questions. We analyse the features and rank them. Our analysis reports that topic entropy, readability of text, metric entropy, and text-code ratio are the top four strong predictors. Our models classify the questions with a maximum of about 76%−87% accuracy that outperforms the accuracy of the state-of-the-art [93] from the literature. We further attempt to predict questions at the time of the question submission. Our machine learning model also predicts questions with exactly the same accuracy as the classification models. In particular, our model predicts the discouraged questions automatically with a maximum of about 76% precision during their submission. Predicting a potentially down-voted question in advance might greatly help one improve the question during its submission.

6.2 Study Methodology

Figure 6.3 shows the schematic diagram of our study. We first collect about 2.5 million questions from SO. We collect questions from four popular programming languages with certain restrictions. Then we apply the quality metrics to assess their quality from different aspects. We then analyse the agreement between subjective and objective quality measures. Finally, we classify the promoted and discouraged questions based on their objective quality using five machine learning models and evaluate the models' performance. We further attempt to predict the questions during their submission time using a machine learning model and report the model's performance.

Table 6.1
Summary of the Study Dataset

Topic	**Promoted Questions**	**Discouraged Questions**	**Total**
C#	550,173	76,181	626,354
Java	585,410	116,268	701,678
Javascript	598,670	114,645	713,315
Python	418,700	67,035	485,735
Total	2,152,953	374,129	**2,527,082**

6.3 Data Collection

To verify the subjective assessment of SO questions with the objective evaluation, we collect a total number of 2,527,082 questions from SO using StackExchange Data API [87]. In particular, we collect questions related to four major programming languages: C#, Java, JavaScript and Python. We choose these questions under certain restrictions - **(a)** each question must have at least one answer to ensure that the question has attracted the attention of the community **(b)** the question has a score not equal to zero, which indicates that it has obtained at least one up or down vote, and **(c)** questions are posted in SO in the year 2017 or earlier which suggests that the questions have been assessed by SO users for a significant time period. Table 6.1 lists the dataset for our study. We categorise the questions into *two* sub-categories according to their score (i.e., upvotes − downvotes). Each of them is classified as either a high-quality (hereafter, *promoted*) or a low-quality (hereafter, *discouraged*) question as follows.

- **Promoted questions**: Questions with a score greater than 0 (i.e., questions with more upvotes).
- **Discouraged questions**: Questions with a score less than 0 (i.e., questions with more downvotes).

Several objective measures (e.g. reusability of code segment) can be applied to the questions that have a code segment within their body. We thus consider such questions that have at least one line of true code. According to our investigation, 1,909,539 out of 2,527,082 (i.e., 75.56%) questions have at least one line of code. We identify such questions and extract code segments using specialised HTML tags such as `<code>` under `<pre>`, and select them for our study. We also collect a total of $54,463$ tags from the *Tags* table using StackExchange Data API [87]. We use this data in calculating topic entropy of a question.

6.4 Metric Selection

A question from SO consists of three components: *title, body,* and *tags*. The title summarises the overall problems, body describes the problem in detail, and tags indicate the relevant topics (e.g., java) of a question. Questions that describe code related issues generally attach a code segment to support the problem

Table 6.2
Summary of the objective metrics

Metric	Sub-metric	Question Parts of Interest	Description
Title Quality	–	Title	Measures the integrity between the title and other parts of a question.
Readability	Text Readability	Body texts	Estimates the quality of the written explanation
	Code Readability	Code segment	Measures code structure and important local features
Explanation Quality	Text-code Ratio	Code segment + Explanation	Measures the comprehensiveness of the code explanation.
	Text-code Correlation	Code segment + Explanation	Measures the coherence between code segments and their explanation.
Code Reusability	Parsability	Code segment	Measures the parsability of the attached code segment.
Code Understandability	No. of API Used	Code segment	Calculates the number of APIs used in the code segment.
Topic Entropy	–	Tags	Estimates the ambiguity of the question topics.
Metric Entropy	–	Body texts	Estimates the randomness of the terms used in the question texts.
Sentiment Polarity	–	Title + Body texts	Determines the positive, negative or neutral sentiment from question text.

statements. To investigate the agreement between subjectivity and objectivity, we thus attempt to assess all of these components with appropriate metrics.

Table 6.2 shows the metric summary. First, we measure the quality of the title by estimating how appropriately the title summarizes the overall problem. Second, we measure the readability of the text and code to check how easy a question's text or code is to understand. Third, we investigate how appropriately the attached code segment is explained by the question submitter. Fourth, the reusability and understandability of the code segments are assessed. Fifth, we calculate topic entropy to check whether a single question touches too many topics or not. Questions containing multiple topics are imprecise or ambiguous, and thus are difficult to answer. Sixth, we measure metric entropy to estimate the randomness of the terms used in the body texts. Finally, we measure the sentiment polarity of the question texts to see whether the emotional style of a technical question does influence its quality or not.

6.5 Answering RQ$_1$: A comparative study between promoted and discouraged questions using objective metrics

To answer RQ$_1$, we analyse both promoted and discouraged questions from ten quality aspects as shown in Table 6.2. In particular, we compare the objective quality metrics of promoted questions to that of discouraged questions. Our goals are to determine (1) whether there exists any noticeable differences between these two set of metrics, as a result, (2) whether they either support or contradict the subjective assessment made by SO users. Finally, we attempt to derive a conclusion from such findings for RQ_1. Our comparative study is divided into several different analyses as follows:

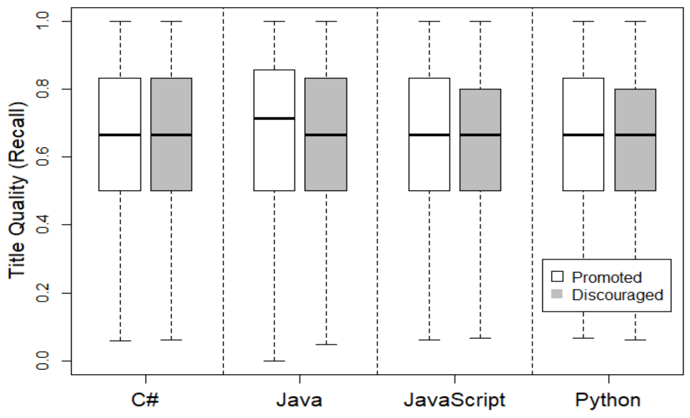
6.5.1 Title Quality (TQ)

The title is an important part of a SO question. Title should properly summarise the question. On the contrary, the body texts explain the programming problem in detail with comprehensive information. If the title is ambiguous or poorly written, the question might fail to draw the attention of the potential users who could have answered the question. We thus measure the title quality using a popular metric namely ROUGE. ROUGE automatically determines the quality of a summary by comparing it with human written summaries. The measure counts the number of overlapping units such as n-gram, word sequences, and word pairs between an auto-generated summary and the ideal summaries created by humans. ROUGE also has a high correlation with human scores [30]. Here, we use ROUGE 2.0 [36], the latest ROUGE tool for our analysis. There are different types of ROUGE measures such as ROUGE-N: *N-gram Co-Occurrence Statistics*, ROUGE-L: *Longest Common Subsequence*, ROUGE-W: *Weighted Longest Common Subsequence*, and ROUGE-S: *Skip-Bigram Co-Occurrence Statistics*, ROUGE-SU: *Skip-Bigram with the addition of unigram as counting unit.* After a careful analysis, we find that only ROUGE-1 (unigram) works reasonably well in our problem context. We consider title of a question as a reference summary (i.e., human generated summary). ROUGE takes the body texts that contain more detailed information about the problem and produces a summary. Such an automatically produced summary is called system summary. Then we calculate recall to determine the quality of title. In particular, we calculate the ratio between the number of overlapping words and total number of words in the reference summary (i.e., question title) as follows.

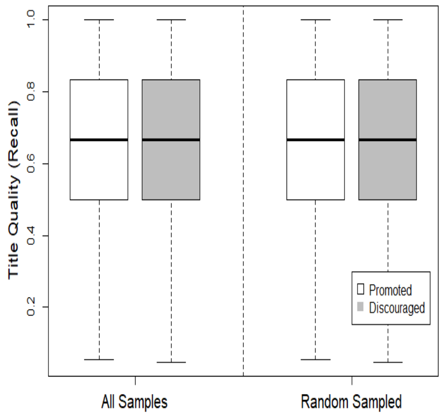
$$Recall = \frac{\mathbf{COUNT}(Reference\ Summary\ \cap\ System\ Summary)}{\mathbf{COUNT}(Words\ in\ Reference\ Summary)} \tag{6.1}$$

Does the title quality agree with subjective evaluation? As shown in Figure 6.4, we do not find any significant differences in the title quality between promoted and discouraged questions. First, we attempt to determine the title quality of both the promoted and discouraged questions. We find that Java titles of promoted questions have slightly higher recall than those of discouraged questions. The other three languages have nearly the same precision for both the question categories. Then we combine the samples of all the languages and also did not notice any significant differences. Finally, we undersampled the promoted questions to balance the dataset and find similar results.

Summary of Title Quality: There is no significant difference in the title quality between promoted and discouraged questions. Both the titles summarize the problem statements fairly well with 60%+ ROUGE scores. Thus, title quality might not be a strong metric for verifying the subjective assessment of the questions from SO.



(a) Title quality for each of the programming languages (*promoted* vs *discouraged*)



(b) Title quality for all the programming languages (*promoted* vs *discouraged*)

Figure 6.4: Title quality

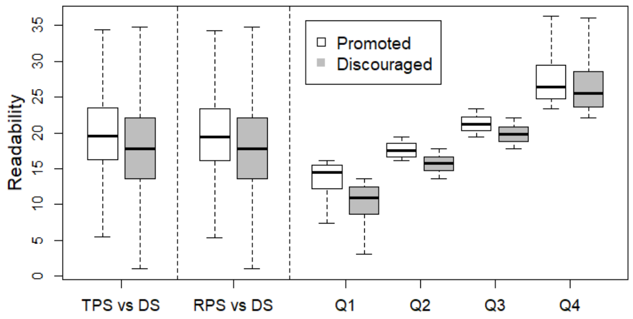
6.5.2 Readability

Readability is the ease with which a reader can understand a written text [138]. Question from SO generally contain two types of items - regular texts and code segments. We thus attempt to capture the readability of both texts and code using two separate metrics: *Text Readability* and *Code Readability*. We extract code segments and texts (including title) by parsing the HTML contents of each question. In our dataset, about 25% of questions have textual description only, and the other 75% contain both the code segments and texts. We apply the readability metrics to them as follows:

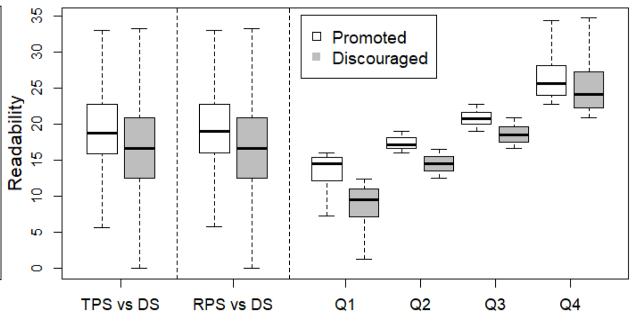
Text Readability (TR)

Readability of texts depends on the complexity of their vocabulary, syntax, and their presentation style such as font size, line height, and line length [138]. We consider the textual description from the question body to measure text readability. We compute the text readability separately for the questions having (1) textual description only and (2) textual description and code segments. To assess the text readability, we compute a popular readability index namely *Readability Index (RIX)* [4]. This readability index represents the comprehension difficulty of passages written in English and are based on US grade levels. First, we calculate the readability of each of the body texts of the questions. We then normalize the score between 0 and 100. A low score indicates the high readability whereas a high score indicates the poor readability of the question texts.

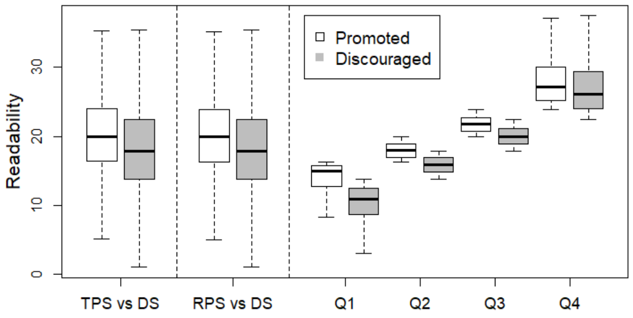
Do the promoted questions have higher text readability than that of the discouraged questions? First, we select only 25% questions that do not have any code segments in their body. As shown in Figure 6.5, we find a significant difference in text readability between promoted and discouraged questions, where promoted questions have higher text readability scores than those of discouraged questions. That is,



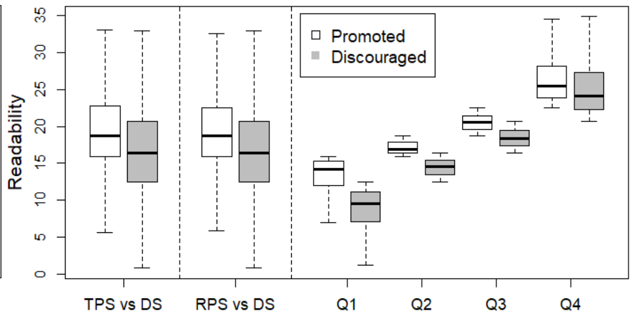
(a) C# questions with no code segment.



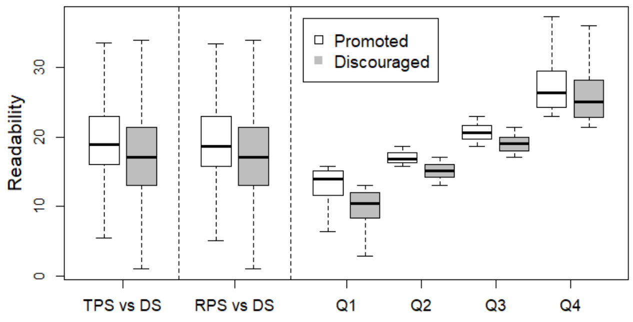
(b) C# questions with code segment.



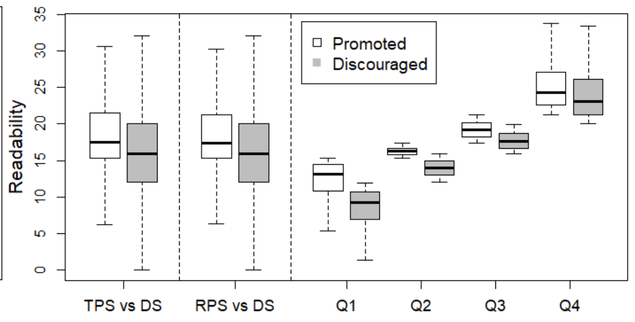
(c) Java questions with no code segment.



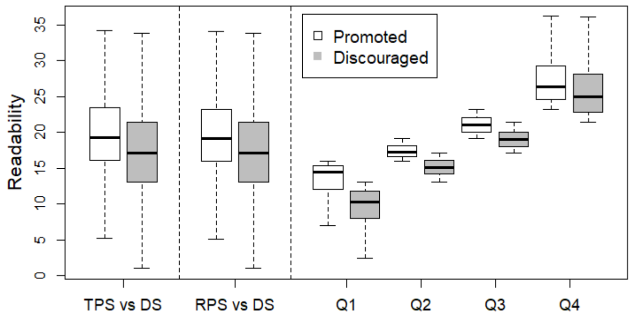
(d) Java questions with code segment.



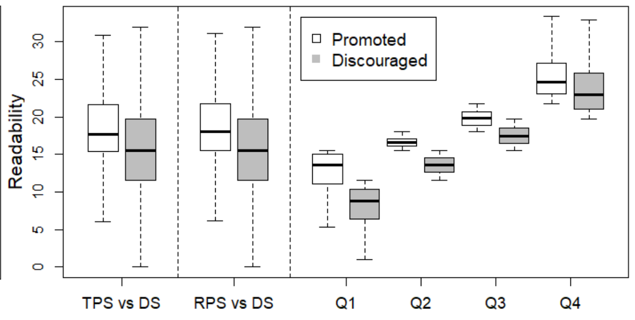
(e) Javascript questions with no code segment.



(f) Javascript questions with code segment.



(g) Python questions with no code segment.



(h) Python questions with code segment.

Figure 6.5: Text readability (**TPS** = Total Promoted Sample, **RPS** = Random Promoted Sample, **DS** = Discouraged Sample, **Q1** = First quartile, **Q2** = Second quartile, **Q3** = Third quartile and **Q4** = Fourth quartile).

promoted questions are less readable than that of the discouraged questions. In our dataset, the number of promoted questions is higher than the discouraged questions. We thus undersampled the promoted questions to balance the dataset. Then we compare the readability of discouraged samples with that of the promoted samples. For all the programming languages, we get significant *p-values* from the *Mann-Whitney-Wilcoxon* statistical test and small *effect-size* from *Cliff's delta* test. For Java, we get *p-value* $= 0 < .05$ and *Cliff's d = 0.20 (small).* We also get similar results for the other three languages - C#, JavaScript and Python. Next, we divide the readability scores into four quartiles and compare them using box plots. We find statically significant differences in text readability (i.e., *p-values* $< .05$) for all the quartiles between promoted and discouraged questions. However, the *effect-sizes* differ from small to large. Second, we measure the readability of the rest 75% questions that have both code segments and textual description. In this case, we also find a significant difference in readability between promoted and discouraged questions.

We further investigate why the promoted questions have lower readability. We find that promoted questions often contain code segments or program elements (e.g., API classes) which are considered as complex words. The existing readability metrics are not well designed for handling the code or program elements within the texts.

Summary of Text Readability: There is a significant difference in text readability between promoted and discouraged questions. Promoted questions have higher readability score than those of the discouraged questions. That is, promoted questions are less readable than discouraged questions. Their readability degrades because they often include code segments or program elements (e.g., API classes).

Code Readability (CR)

Code readability is a property that determines how easily a given code snippet can be read and understood by the developers who did not author the code [95]. Code readability is strongly related to software maintainability and quality. Code understanding is one of the most frequent activities in software maintenance that consumes over 70% of the total maintenance costs [11, 103]. Thus, the underlying idea is - the more readable the code is, the easier it is to reuse and maintain in the long run. Since poor readability of code costs development time and efforts, we analyse the readability of the code segments added to the questions. In particular, we attempt to find whether there is any noticeable difference in the code readability between promoted and discouraged questions.

To compute the code readability, we use the tool developed by Buse and Weimer [17]. Their readability model is trained on human perception of readability or understandability. The model uses several textual features from the code that are likely to affect the humans' perception of readability, and predicts a readability score on a scale from zero to one. Here, one indicates that the source code is highly readable. On the contrary, zero indicates poor readability of the code under analysis.

As shown in Figure 6.6, readability of the code segments found in either promoted or discouraged questions

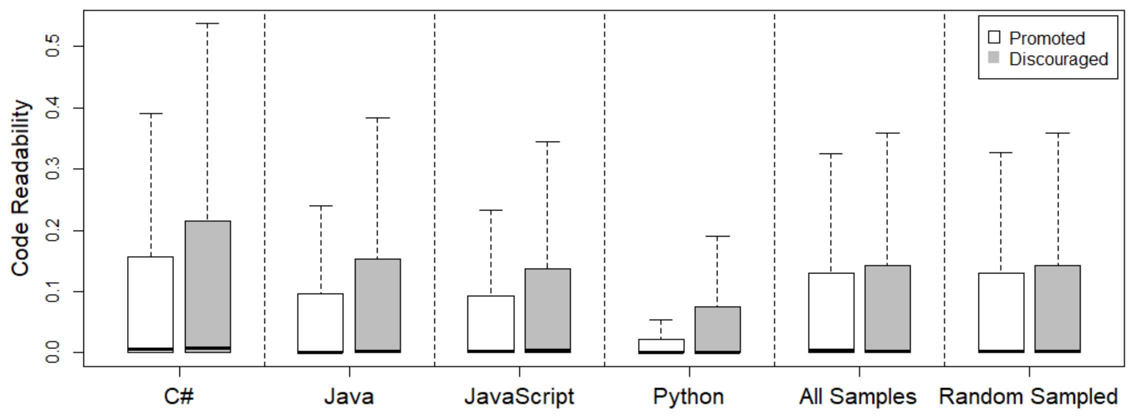


Figure 6.6: Code readability

is very poor.

Do the code examples of promoted questions have a higher readability score than those of discouraged samples? As mentioned above, the average code readability score is very low for the code segments extracted from both promoted and discouraged questions. A significant fraction (34%) of the code segments have a readability score of zero. Only about 14% of code segments have medium or above readability score. Although the *p-values* from *Mann-Whitney-Wilcoxon* statistical test are significant (i.e. *p-values*<.05), the *d-values* from *Cliff's Delta* test show that the *effect sizes* are negligible (i.e., *d <0.1*) for all four cases.

Summary of Code Readability: Readability of the code segments posted on SO questions is generally poor. There is no significant difference in the readability of the code segments between promoted and discouraged questions. Although the code readability score of discouraged questions is marginally higher, the effect size is negligible.

We further manually investigate 200 code snippets (100 promoted + 100 discouraged) from all four programming languages, and make the following observations.

- Most of the code segments are incomplete and they do not maintain proper structures, indentations and naming conventions of identifiers.

- Question submitters often include multiple code segments in a question. When we merge the segments into a single file, they become large and chaotic. Our adopted tool might not be well suited for such synthesized code snippets.

- Many of the code segments are noisy. Programmers often include non-code elements (e.g., stack traces) within the code segment. Such noise might have hurt their readability.

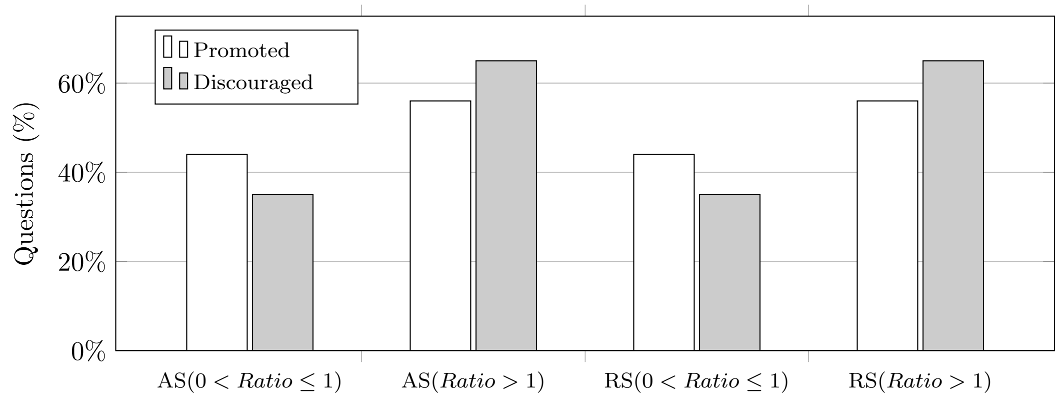


Figure 6.7: Text-code ratio (**AS**=All Samples, **RS**=Random Sampled)

6.5.3 Quality of Explanation for the Code Segments

In SO questions, users often explain their code segments with an associated prose written in regular texts. Ercan et al. [30] suggest that large and unexplained code segments make a question difficult to understand, which might impact the time required to obtain an appropriate answer. An explanation that properly explains the code might help the users to answer the question. Thus, quality of the code explanation might be considered as a metric of question quality. We measure the explanation quality in two ways as follows.

Table 6.3
Summary of the Text-code Ratio (TCR)

Ratio	C#		Java		JavaScript		Python	
	Promoted	Discouraged	Promoted	Discouraged	Promoted	Discouraged	Promoted	Discouraged
Ratio <= 1	46 %	36 %	39 %	28 %	44 %	35 %	51 %	44 %
Ratio >1	54 %	64 %	61 %	72 %	56 %	65 %	49 %	56 %

Text-code Ratio (TCR)

The metric determines how extensively the code has been explained by the associated prose in a question. To examine how comprehensive the code explanation is, we extract the code segments and texts from the body of a question. Then we calculate their lengths in terms of number of ASCII characters they used. Finally, we measure the text-code ratio as follows:

$TCR = \frac{\textbf{LENGTH}(code)}{\textbf{LENGTH}(texts)}$

Do promoted questions contain more explanation for their code than that of the discouraged questions? We divide the text-code ratio (TCR) into two categories: (1) $0 < Ratio \leq 1$ and (2) $Ratio > 1$. When the length of the texts is larger or equal to the code, these questions belong to the first category. The second category includes the questions which have a greater code length than that of the texts. From Figure 6.7, we see that the promoted questions have more code explanations than the discouraged questions. Squire and Funkhouser [110] suggest that high-quality questions have a more detailed explanation of code segments than that of low-quality questions. As shown in Table 6.3, all the promoted questions of category 1 have a higher code explanation ratio than that of discouraged questions from the four programming languages. For example, promoted questions of C#, Java, JavaScript, and Python have 10%, 11%, 9%, and 7% higher text-code ratios respectively than that of discouraged questions.

We also determine whether the difference of text-code ratio between promoted and discouraged questions is statistically significant. We use the *Mann-Whitney-Wilcoxon* statistical test and find significant p-value *(p-value = 0)* with a *small* effect size.

Summary of Text-code Ratio: According to our investigation, promoted questions have a more detailed explanation for their code than that of the discouraged questions. The ratio difference is statistically significant with a small effect size.

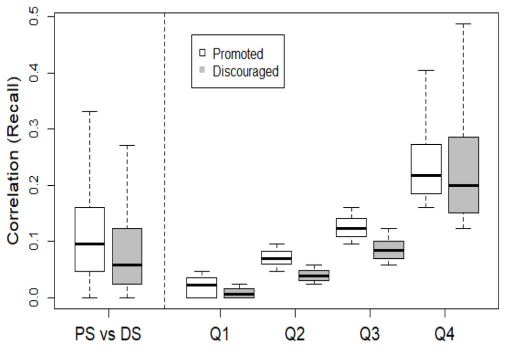
Text-Code Correlation (TCC)

Table 6.4
Summary of Statistical Significance Tests of Text-code Correlation (TCC)

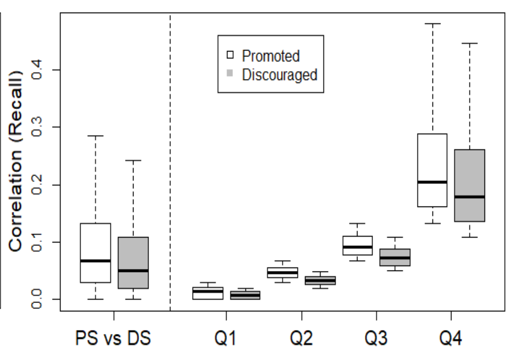
Dataset	Test Name	C#	Java	JavaScript	Python	All Languages
First Quartile	MWW test (p-value)	0	0	0	0	0
	Cliff's d (effect size)	0.41 (medium)	0.29 (small)	0.0.35 (medium)	0.21 (small)	0.32 (small)
Second Quartile	MWW test (p-value)	0	0	0	0	0
	Cliff's d (effect size)	0.93 (large)	0.65 (large)	0.69 (large)	0.07 (negligible)	0.61 (large)
Third Quartile	MWW test (p-value)	0	0	0	0	0
	Cliff's d (effect size)	0.85 (large)	0.55 (large)	0.59 (large)	-0.08 (negligible)	0.45 (medium)
Fourth Quartile	MWW test (p-value)	0	0	0	0	0
	Cliff's d (effect size)	0.19 (small)	0.21 (small)	0.19 (small)	-0.10 (negligible)	0.11 (negligible)
All Samples	MWW test (p-value)	0	0	0	0	0
	Cliff's d (effect size)	0.22 (small)	0.13 (negligible)	0.15 (small)	0.10 (negligible)	0.11 (negligible)

The previous section measures the comprehensiveness of the code explanation. In this section, we determine the appropriateness of the code explanation by measuring the correlation between code segments and code explanation within a question. Ercan et al. [30] evaluate the quality of explanation of a code segment in terms of human readability and suggest that a high textual overlap between code segments and code explanation is strongly correlated to the quality of a question. We thus investigate the content overlapping between code explanations and code segments using Recall-Oriented Understudy for Gisting Evaluation (ROUGE). In particular, we use code segments as a reference summary and the automatically produced summary by ROUGE from code explanations as a system summary. Then we calculate recall base on the ratio between the number of overlapping words and total words in the reference summary. The resulting ROUGE scores (i.e., recall) indicate how appropriately the code explanation (system summary) explain the target code (reference summary).

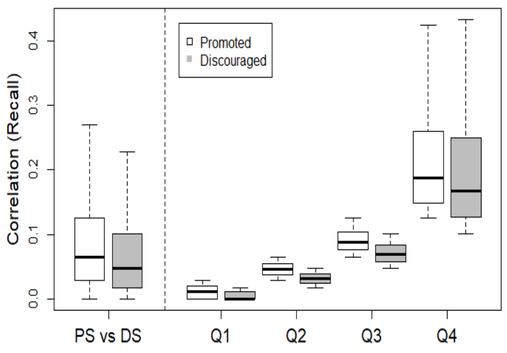
Do the promoted questions have a higher correlation between their code segments and code



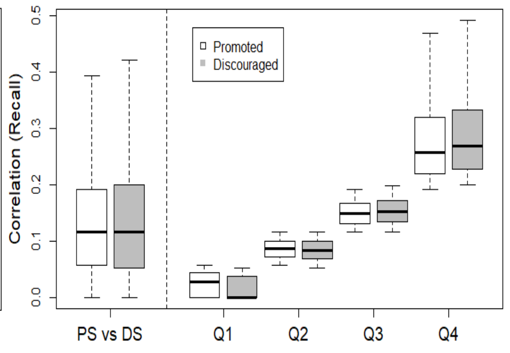
(a) C# programming questions



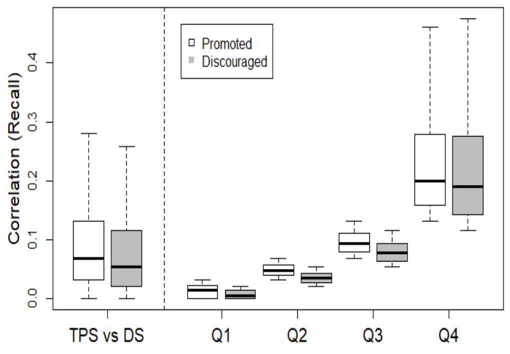
(b) Java programming questions



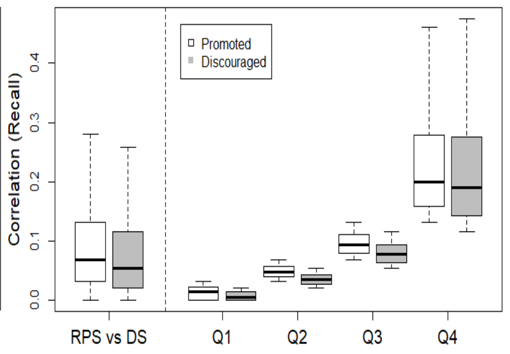
(c) Javascript programming questions



(d) Python programming questions



(e) All programming languages



(f) All programming laguages (After Random Sampling)

Figure 6.8: Correlation measurement using ROUGE-2.0 (**TPS** = Total Promoted Sample, **RPS** = Random Promoted Sample, **DS** = Discouraged Sample, **Q1** = First quartile, **Q2** = Second quartile, **Q3** = Third quartile and **Q4** = Fourth quartile).

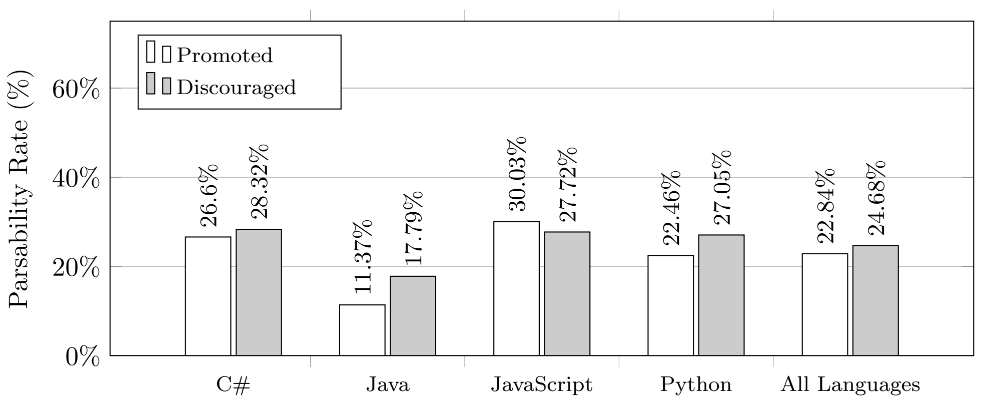


Figure 6.9: Parsability rates of the programming languages (promoted VS discouraged questions).

explanations than those of the discouraged questions? According to our analysis, the correlation between code segments and code explanations of promoted questions is stronger than that of the discouraged questions. Figure 6.8 summarizes the results as box plots. We see that all the promoted questions have a significantly higher correlation between explanation and code with small to large effect sizes.

We use *Mann-Whitney-Wilcoxon* statistical test and *Cliff's delta* test to compute the statistical difference and effect size respectively. The result of the statistical test is shown in Table 6.4. We see that the first, second, and third quartile have a statistically significant difference with high effect size, whereas the fourth quartile shows small or negligible effect size.

Summary of Text-code Correlation: Overall, the text-code correlation is higher for the promoted questions than the discouraged questions. So, we can conclude that this objective evaluation agrees with the subjective measure. For several reasons, Java has an opposite result among the four programming languages.

6.5.4 Code Reusability (CRUSE)

Usability of source code is determined based on parsability, compilability and executability of source code [147]. In this research, we examine only *parsability* as a part of the reusability of code segments found in SO questions. Our goal is to measure the parsability rates of the code segments of four programming languages (C#, Java, JavaScript, and Python). We then contrast the parsability rate of promoted questions to the discouraged questions. For parsing C# code segments, we use a tool called *Roslyn* by Microsoft. Roslyn provides rich APIs (e.g., CSharpSyntaxTree.ParseText) for parsing the abstract syntax tree from the C# code. To parse Java code segments, we used *JavaParser*[3]. *Esprima*[4] was used to parse JavaScript code examples. Python's built-in AST module was used to parse the Python code segments.

[3]http://javaparser.org
[4]http://esprima.org

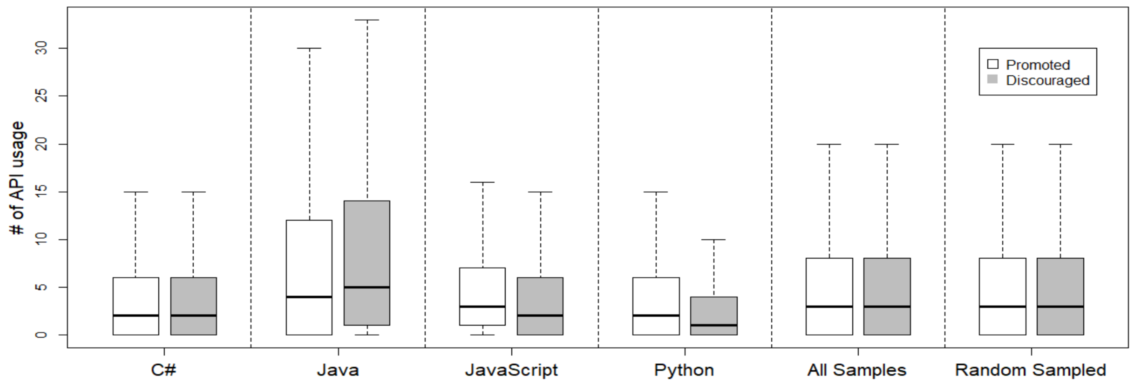


Figure 6.10: Code understandability

Are code segments from promoted questions easier to parse than those of the discouraged questions? As shown in Figure 6.9, the parsability rate is higher for the code examples from the discouraged questions than that of the promoted questions. Such an observation is true for three out of four programming languages - C#, Java and Python. We find that the overall parsability rate of code examples from promoted questions is about 23% whereas such ratio is 25% for the code segments of discouraged questions. We further attempt to find why code examples of the discouraged questions have higher parsing rates through manual analysis. We make several observations that might explain such an inconsistency as follows.

- **Redundant and long code:** The users at SO often submit questions that contain unnecessarily long code. Such questions often receive down-votes and fail to get appropriate answers even though their code is parsable. According to our investigation, such code segments have more than 72 lines of code. We believe that such long code places extra burden on the other users during question answering. Thus, despite being parsable without modifications, such questions receive down-votes and considered as discouraged ones.

- **Novice user:** Sometimes novice users add long code (e.g. an entire course assignment) without proper analysis. They even do not discuss about the encountered issues properly. According to our analysis, 50% of users who added long and redundant code and receive down-votes have a reputation score below 20.

Summary of Reusability: Code examples of promoted questions have a little lower parsing rate than those of the discouraged code examples.Thus, the rate of reusability (i.e., parsability) marginally disagrees with subjective measures.

6.5.5 Code Understandability (CUA)

The code fragments on SO posts are often accompanied by natural language texts. We already measured the quality of code explanation using such texts. We also estimate the readability of the code segments using an existing tool [17]. However, the perceived readability is something different from the actual understandability of the code. A developer could find a piece of code readable, but still difficult to understand what the code does [104]. Scalabrino et al. [104] report that code understandability often depends on the number of APIs used in a program. The more APIs are used, the harder would be the code to comprehend. Thus, we count the number of APIs in the code segments as a proxy of code understandability. We use regular expressions to find the APIs used within the code segments.

Are code segments from promoted questions more understandable than those of discouraged questions? According to our analysis, the difference in the number of APIs usage between promoted and discouraged questions is not statistically significant as shown in Figure 6.10. We further investigate the questions from each of the programming languages and find the followings:

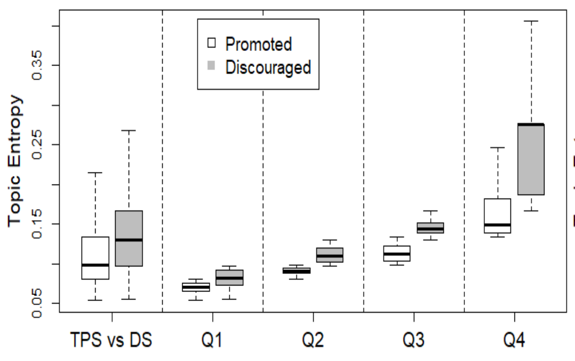
C#. About 50% of the promoted questions use at least two API classes whereas 25% of them use at least six API classes in their code. The discouraged questions have almost similar statistics with a maximum count of 276 API classes. On the contrary, promoted questions might even contain a total of 314 API classes in their code snippets.

Java. About 50% of the promoted questions use at least four API classes whereas 25% of them use at least twelve API classes in their code. On the other hand, about 50% of the discouraged questions use at least five API classes whereas 25% of them use at least fourteen API classes in their code. The discouraged questions have a maximum count of 817 API classes. On the contrary, promoted questions might even contain a total of 604 API classes in their code snippets.

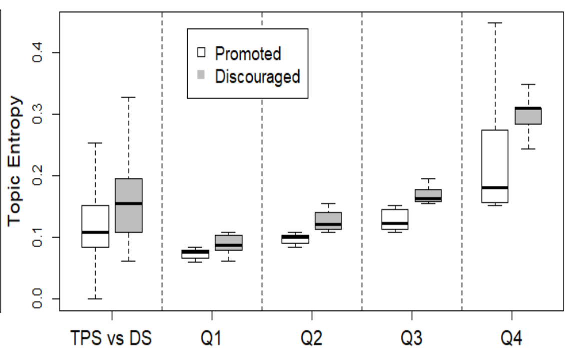
JavaScript. About 50% of the promoted questions use at least three API classes whereas 25% of them use at least seven API classes in their code. On the other hand, about 50% of the discouraged questions use at least two API classes whereas 25% of them use at least six API classes in their code. The discouraged questions have a maximum count of 616 API classes. On the contrary, promoted questions might even contain a total of 586 API classes in their code snippets.

Python. About 50% of the promoted questions use at least two API classes whereas 25% of them use at least six API classes in their code. On the other hand, about 50% of the discouraged questions use at least one API classes whereas 25% of them use at least four API classes in their code. The discouraged questions have a maximum count of 440 API classes. On the contrary, promoted questions might even contain a total of 430 API classes in their code snippets.

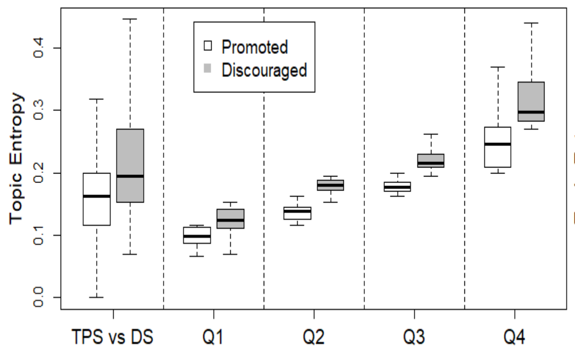
Summary of Code Understandability: Dynamically typed languages such as JavaScript and Python have more APIs in the code examples of their promoted questions than that of the discouraged questions. On the other hand, statically typed language as Java has less APIs in the code examples of their promoted questions. Thus, promoted code snippets from statically typed languages are more readable than the discouraged code snippets. However, the opposite is found to be true for dynamically typed languages.



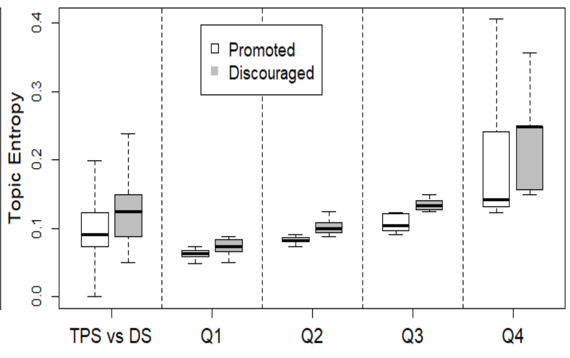
(a) C# programming questions



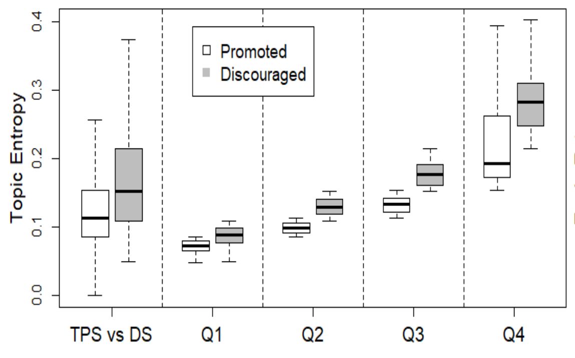
(b) Java programming questions



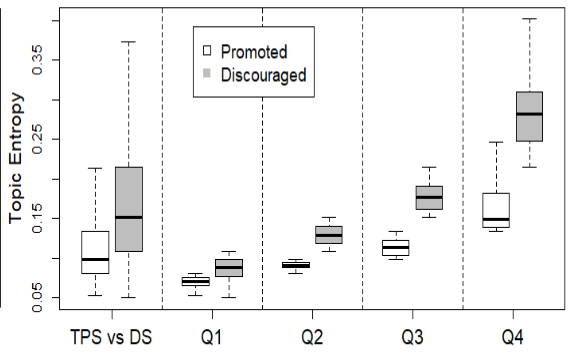
(c) Javascript programming questions



(d) Python programming questions



(e) All programming languages



(f) All programming laguages (After Random Sampling)

Figure 6.11: Summary of topic entropy (**PS** = Promoted Sample, **TPS** = Total Promoted Sample, **RPS** = Random Promoted Sample, **DS** = Discouraged Sample, **Q1** = First quartile, **Q2** = Second quartile, **Q3** = Third quartile and **Q4** = Fourth quartile).

Table 6.5
Summary of Statistical Significance Tests of Topic Entropy (TE)

Dataset	**Statistical Significance Test**	**C#**	**Java**	**JavaScript**	**Python**
First Quartile	MWW test (p-value)	0	0	0	0
	Cliff's d (effect size)	-0.61 (large)	-0.66 (large)	-0.68 (large)	-0.60 (large)
Second Quartile	MWW test (p-value)	0	0	0	0
	Cliff's d (effect size)	-0.99 (large)	-0.99 (large)	-0.98 (large)	-0.96 (large)
Third Quartile	MWW test (p-value)	0	0	0	0
	Cliff's d (effect size)	-0.99 (large)	-1.0 (large)	-0.99 (large)	-1.0 (large)
Fourth Quartile	MWW test (p-value)	0	0	0	0
	Cliff's d (effect size)	-0.74 (large)	-0.79 (large)	-0.68 (large)	-0.74 (large)
All Samples	MWW test (p-value)	0	0	0	0
	Cliff's d (effect size)	-0.34 (medium)	-0.36 (medium)	-0.35 (medium)	-0.33 (small)

6.5.6 Topic Entropy (TE)

Programming questions discussing on only common topics (e.g., java) are likely to be imprecise or ambiguous. Precise topics could help the users locate the questions of their interest. We study the uncertainty or ambiguity of question topics and attempt to determine if discouraged questions use more common topics than those of promoted questions. In SO, tags capture the topics with which a question is associated [150]. Each question can have at most five tags and must have at least one tag.

In information theory, entropy is considered as a measure of uncertainty of a random variable that takes up different values [97]. To measure the topic entropy associated with each question, we first calculate the probability of each topic (i.e., tag). Consider the topic probability of a particular topic i is P_i.

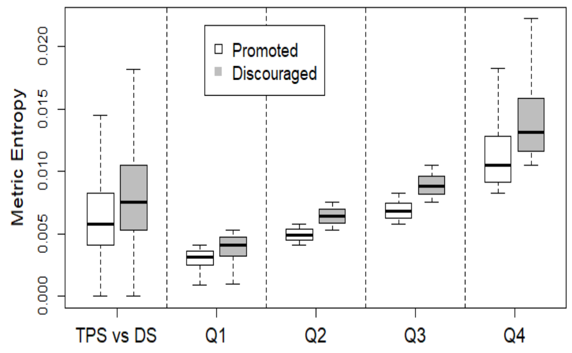
$$P_i = \frac{F_i}{\sum_{i=1}^{N} F_i} \tag{6.2}$$

where F_i is the frequency count of the i^{th} topic and N denotes the total number of topics. We then determine topic entropy for each question as follows:

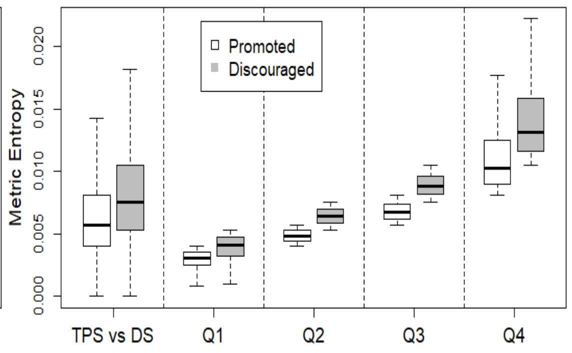
$$TE = -\frac{1}{n}\left[\sum_{k=1}^{n} P_k \times log(P_k)\right] \tag{6.3}$$

n denotes the total number of topics (i.e., tags) of a question. Here, we calculate the average entropy (dividing the entropy by n) since the number of tags differs from question to question. Then we normalize them. If a question of SO uses a set of fairly common topics, the entropy becomes higher. According to information theory, a higher entropy value indicates high ambiguity or uncertainty of question topics and vice versa.

Do the promoted questions have less ambiguous topics? Figure 6.11 shows the box plots of topic entropy. We see that topic entropy of promoted questions is lower than that of the discouraged questions for each of the programming languages. In other words, promoted questions have precise topics whereas the topics of the discouraged questions are ambiguous. We further test whether the differences of topic entropy



(a) All programming languages



(b) All programming laguages (After Random Sampling)

Figure 6.12: Summary of metric entropy (**PS** = Promoted Sample, **TPS** = Total Promoted Sample, **RPS** = Random Promoted Sample, **DS** = Discouraged Sample, **Q1** = First quartile, **Q2** = Second quartile, **Q3** = Third quartile and **Q4** = Fourth quartile).

are statistically significant for each of the languages. Table 6.5 shows the test results. We use the *Mann-Whitney-Wilcoxon* test and find significant p-values (i.e., *p-value* = *0*) for all the test cases. Our quartile analysis using *Cliff's delta* test suggests that the differences are significant with a large effect size for all four quartiles. The difference is also statistically significant with medium or small effect size when we take the entire dataset.

Summary of Topic Entropy: Topic entropy of promoted questions is lower than that of discouraged questions. Thus, topics of promoted questions are more precise than that of discouraged questions.

6.5.7 Metric Entropy (ME)

We use metric entropy to contrast between promoted and discouraged questions. Metric entropy is the Shannon entropy [123] divided by the length of the text. It represents the randomness of the information contained in a question text [93]. A higher metric entropy indicates that the questions have used the common words frequently in its texts and vice versa.

Do the promoted questions have used more unusual terms? Figure 6.12 shows the box plots of metric entropy. We see that promoted questions have lower metric entropy than that of the discourage questions for all the programming languages. That is, promoted questions use more unusual terms (e.g., more technical terms) whereas the terms of the discouraged questions are common. We then use *Mann-Whitney-Wilcoxon* test and find significant p-values (i.e., $p = 0 < .05$) for the entire dataset. However, the effect-size is small. Our quartile analysis using *Cliff's delta* test suggests that the differences are significant with a medium or large effect size for all four quartiles.

Summary of Metric Entropy: Metric entropy of promoted questions is lower than that of discouraged questions. Thus, terms of promoted questions are more unusual than that of discouraged questions.

Table 6.6
Details of Sentiment Polarity

Language	**Question Category**	**Sentiment Polarity (in-percent)**			
		Positive	**Negative**	**Mixed**	**Neutral**
Java	Promoted	23.60	17.26	15.46	43.68
	Discouraged	20.95	16.25	11.05	51.74
C#	Promoted	24.03	16.78	15.53	43.66
	Discouraged	22.00	15.05	10.00	52.94
Javascript	Promoted	26.53	14.71	14.75	44.00
	Discouraged	24.52	12.71	9.94	52.81
Python	Promoted	23.48	17.47	15.77	43.29
	Discouraged	21.43	16.70	11.33	50.53

6.5.8 Sentiment Polarity (SP)

Software developers often interact with each other in various community-based Q&A sites (e.g., SO) and development environments (e.g., Github). Calefato et al. [18] argued that social aspects such as emotional status of a developer might have an impact on the response time in Q&A site. That is, sentiment expressed in the question texts might determine its answering time of the question. We thus consider the sentiment polarity of SO question texts as an objective quality aspect. We used *SentiStrength-SE*, a sentiment analysis tool to measure the sentiment polarity. SentiStrength-SE is developed by Islam and Zibran [51] for the software engineering domain.

Does sentiments expressed in SO question influence their subjective assessment? We measure *positive, negative, mixed* and *neutral* sentiment polarity of the texts extracted from the title and body of a question. Table 6.6 shows the summary of sentiment measures of the SO questions. We see that about half of the contents from both promoted and discouraged questions are neither positive nor negative. We also see that texts of promoted questions have a higher percentage of positive sentiment and a lower percentage of neutral sentiment polarity than those of discouraged questions. However, none of the difference is statistically significant.

Summary of Sentiment Polarity: Results show that sentiment polarity has less impact on the subjective quality assessment of SO questions. About 50% of the questions have neither positive nor negative sentiment. Although, more questions are having positive sentiment than negative sentiment, the difference between such ratios is not statistically significant.

Table 6.7
Range, Central Tendency, and Standard Deviation

Attributes	Promoted Questions					Discouraged Questions				
	Min	Max	Mean	Median	Std. Dev.	Min	Max	Mean	Median	Std. Dev.
Title Quality	0	1	0.68	0.67	0.24	0	1	0.64	0.67	0.26
Text Readability	0	100	19.85	19.15	5.71	0	100	16.86	16.46	6.85
Code Readability	0	1	0.15	0.003	0.28	0	1	0.15	0.003	0.28
Text-code Ratio	0	2660	2.65	1.20	7.14	0	5995	3.93	1.65	20.72
Text-code Correlation	0	1	0.10	0.07	0.11	0	1	0.09	0.05	0.12
No. of API Used	0	604	7.06	3	14.11	0	817	7.34	3	16.40
Topic Entropy	0	1.0	0.13	0.11	0.06	0	1.0	0.17	0.15	0.08
Metric Entropy	0	0.04	0.006	0..006	0.004	0	0.04	0.008	0.008	0.004

Table 6.8
Feature Ranks (Information Gain)

Rank	Feature	Score
1	Topic Entropy	0.6931
2	Metric Entropy	0.5328
3	Text-code Ratio	0.3156
4	Code Readability	0.2803
5	Text Readability	0.2486
6	Text-code Correlation	0.0167
7	Title Quality	0.0044
8	Sentiment Polarity	0.0040
9	Code Understandability	0.0021
10	Code Reusability	0.0017

Table 6.9
Feature Analysis using Decision Trees

Rank	Feature	Accuracy
1	Topic Entropy	66.36%
2	Text Readability	63.44%
3	Metric Entropy	60.49%
4	Text-code Ratio	56.25%
5	Text-code Correlation	54.11%
6	Sentiment Polarity	53.95%
7	Code Understandability	53.01%
8	Title Quality	52.86%
9	Code Readability	52.18%
10	Code Reusability	52.17%

6.6 Answering RQ$_2$: Analysis, Classification, and Prediction of Questions base on their Quality

In this section, we present the methodologies towards understanding the characteristics of promoted and discouraged questions. Next, we describe the machine learning models with their settings. We then feed the models with the features described at section 6.5, and evaluate their performance. Finally, we compare our models' performances with the baseline models.

6.6.1 Range, Central Tendency, and Standard Deviation

In order to investigate the characteristics of promoted and discouraged questions, we measure the ranges (e.g., max, min), central tendencies (e.g., mean, median), and standard deviations of the attributes (Table 6.7) extracted from both the question classes. We find that the arithmetic mean of three attributes (e.g., title quality, text readability, text-code correlation) of promoted questions are greater than those of the discouraged questions. That is, titles of promoted questions summarize their description more appropriately. Promoted questions also describe the code segments more than those of the discouraged questions. But the description texts of promoted questions are harder to read than those of the discouraged questions. According to our analysis, code elements within the texts of promoted questions hurt the readability. On the other hand, four attributes (e.g., topic entropy, metric entropy, no. of API used, text-code ratio) of promoted questions has a lower arithmetic mean than discouraged questions. It means that promoted questions are less ambiguous, use less common terms, easy to understand, and well explained. Except for code readability, standard deviations of all the attributes from the promoted samples are smaller than those of the discouraged samples. It means that the attributes of promoted samples are close to the mean, on the contrary, the attribute values of discouraged samples are farther away from the mean, on average. We exclude parsability and sentiment polarity from this type of measurement because they have either nominal or categorical values.

6.6.2 Ranking Features

In the previous section, we showed that there are several attributes, whose values are different for promoted and discouraged questions. Such findings indicate that these attributes might be the key estimators in predicting whether a question will be up-voted or down-voted. However, we do not know yet which attributes are more important than others in differentiating up-voted and down-voted questions. Therefore, a ranking of these attributes might help to select the top N estimators in predicting questions. Learning the appropriate values from millions of questions in a high dimensional space is computationally expensive. Thus, reducing the number of attributes is important. We use the popular statistical measures such as information gain based ranking algorithm to rank our attributes as follows.

Information Gain: We attempt to determine which attributes are stronger than others for discriminating promoted and discouraged questions. We thus employ information gain based feature ranking technique because it can estimate the discrimination power of each of the given attributes. In information theory, the information gain of a random variable is the change in information entropy between a prior state and a state that takes some information [102]. Therefore, the information gain of a particular attribute in classifying either the questions will be promoted or discouraged is as follows.

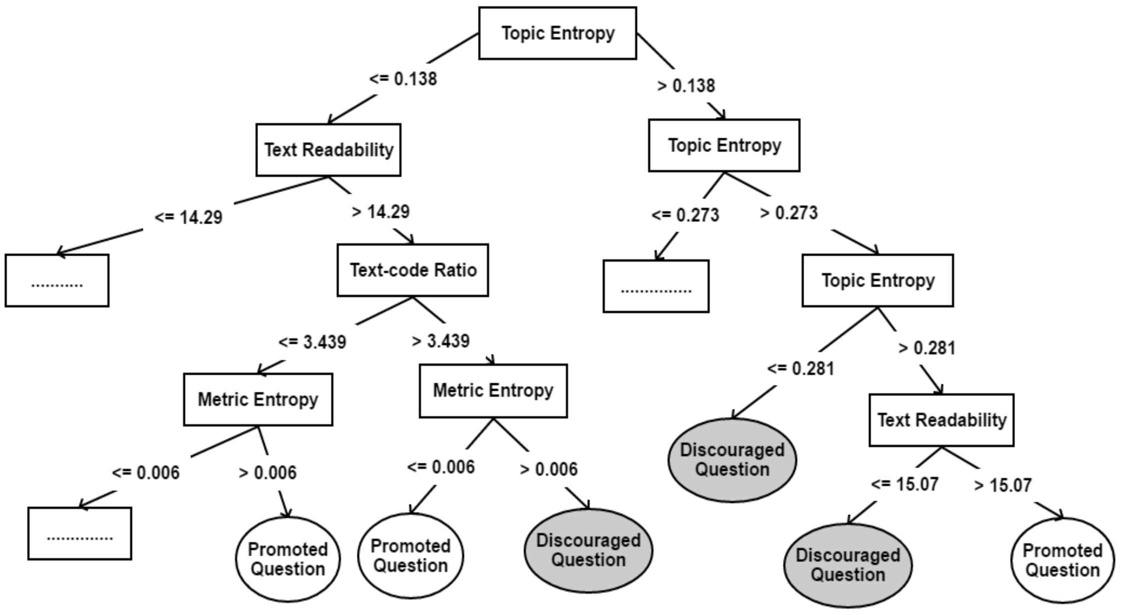


Figure 6.13: Portion of a decision tree trained on our dataset.

$$InfoGain(C, a_i) = H(C) - H(C|a_i) \tag{6.4}$$

where C represents a particular class (i.e., promoted or discouraged), a_i denotes the attribute, and H denotes information entropy. As shown in Table 6.8, topic entropy has the highest information gain. That is, topic entropy might discriminate the questions with the maximum accuracy. Metric entropy, text-code ratio, code readability, and text readability have the second, third, fourth and fifth highest information gain respectively. However, gain ratio of the remaining attributes is either small or negligible. Next, we analyze the discrimination power of each of the attributes using decision trees.

Analysis with Decision Trees: In this study, we not only attempt to predict the quality of a question during question submission time but also to understand which attributes influence the quality of a question. For this reason, we use Decision Trees, a machine learning classification technique whose output can be easily interpreted. A decision tree is a tree in which each internal node (non-leaf) is labelled with a feature, and arcs from any internal node are labelled with exclusive predicates that summarize possible distinct values for the feature [93]. Each leaf of the tree is labelled with a class. As an example, we show a portion of our decision tree (Figure 6.13) trained on our balanced dataset. As we can see, internal nodes represent the quality assessment metrics (e.g., topic entropy). The class (i.e., leaf node) represents the quality to be assigned to the questions exhibiting the conjunction of predicates represented by arcs connecting the root to the leaf. Then we attempt to see how each attribute can classify the promoted and discouraged questions. Table 6.9 shows the classification accuracy of each of the features. We see that topic entropy can classify the questions with overall 66.36% accuracy. Text readability, metric entropy, and text-code ratio have the second, third, and fourth highest discrimination power respectively. As opposed to information gain, we find that text-code correlation has a higher classification accuracy than code readability. However, we see that four of the top five attributes are common in both of the cases. Thus, we select these four features (e.g., topic entropy, text readability, metric entropy, code-text ratio) as an optimal feature set, and analyse the models' performance using all the features and the top four separately.

6.6.3 Classification Models and Settings

According to our comparative study, the relationship between question classes and their corresponding feature values might be complex. Thus, we choose five popular machine learning classification techniques such as – Decision Trees (DT), Random Forest (RF), K-Nearest Neighbors(KNN), Artificial Neural Networks (ANN), Naive Bayes (NB) – that have different learning strategies to estimate the quality of questions. We use *Scikit-learn* [90], one of the popular and widely used tools to implement the techniques.

Parameter Tuning. Tuning parameters in classifiers is important because it changes the heuristics determining how they learn [21]. For example, it controls the number of decision trees to use in Random Forest or the number of clusters in K-Nearest Neighbors (KNN). Models trained with suboptimal parameter settings may underperform as parameter settings depend on the dataset [40]. To select the best model configuration, we use *GridSearchCV*, the cross-validated grid search algorithm of Scikit-learn [90]. The GridSearchCV algorithm works by running and evaluating the model performance of all possible combinations of parameters provided to it. We only tuned significant parameters for each of the models. We choose a number of plausible options for tuning the important parameters of the machine learning models. GridSearchCV does two tasks in parallel (1) cross-validation and (2) parameter tuning. Finally, it returns the values for a model's parameters that maximize the accuracy of the model. Here, we keep the default values for the optional parameters. We also investigate the efficiency of both training and test split set and choose the parameters thereby to avoid model overfitting.

6.6.4 Metrics of Success

When learning from imbalanced datasets (i.e., number of promoted samples are different from discouraged samples), the evaluation of classification performance must be carried out using specific metrics to consider the class distribution and properly assess the effectiveness of learning algorithms [21]. Concerning a reliable assessment of the classification performance of prediction models, the selection of the evaluation criteria is an important part. In a binary classification problem like question quality prediction, a confusion matrix records the results of correctly and incorrectly recognized examples of each class. We can obtain several metrics from the given confusion matrix for the classification performance of both positive and negative classes independently. We thus review several performance metrics and select the most appropriate ones.

The machine learning community often measures the prediction *accuracy* as a simple scalar performance metrics for binary classification. However, accuracy is not capable of providing sufficient information about the performance of a classifier when the dataset is imbalanced [42, 96]. According to He and Garcia [44], accuracy might lead to an incorrect conclusion as the measure is highly sensitive to changes in data. In such cases, *precision* is a useful metric to capture the effect on a classifier performance of having a larger number of negative examples [27]. He and Garcia [44] argued that precision is still sensitive to changes in the data distribution and it cannot assert how many positive examples are classified incorrectly. Unlike precision, *recall* is not sensitive to data distribution. However, any assessment based solely on recall would be inadequate, as it provides no insight into how many examples are incorrectly classified as positives. Neither precision nor recall can provide a reliable assessment of classification performance [21]. These individual scalar metrics can be combined to build more reliable measures of classification performance, particularly in imbalanced situations. Specifically, these aggregated metrics of performance include F-measure that represents the harmonic mean of precision and recall.

Our dataset is imbalanced. The promoted questions are larger than the discouraged questions as shown in section 6.3. Imbalanced dataset might generate suboptimal classification models that provide good coverage of the majority of examples, whereas the minority ones are frequently misclassified [44]. We thus measure precision, recall, F1-score, and overall accuracy to evaluate the model performance so that we can provide a better conclusion on the model performances. In addition, we balance our dataset by under-sampling the promoted questions. Then, we apply the classifiers on the balanced dataset and report the results.

Table 6.10
Performance of Classifiers (**H** indicates the highest values.)

Classifier	**Features**	**Dataset**	**Promoted Question**			**Discouraged Question**			**Overall**
			Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
DT	**All**	**Imbalanced**	90.90	89.98	90.44	59.27	61.80	60.51 **H**	84.60
		Balanced	74.56	74.17	74.37	74.30	74.70	74.50	74.30
	Top-4	**Imbalanced**	91.85	91.10	91.48 **H**	63.51	65.73	64.60 **H**	86.26 **H**
		Balanced	75.37	74.90	75.14 **H**	75.06	75.53	75.29 **H**	75.21 **H**
RF	**All**	**Imbalanced**	88.39	94.03	91.12 **H**	65.29	47.65	55.09	85.18 **H**
		Balanced	76.98	72.09	74.46	73.76	78.44	76.03 **H**	75.27
	Top-4	**Imbalanced**	89.01	94.45	91.45	68.22	50.54	58.06	86.07
		Balanced	75.17	71.90	73.50	73.07	76.25	74.62	74.07
ANN	**All**	**Imbalanced**	83.01	97.45	89.65	58.79	15.40	24.41	81.80
		Balanced	73.54	79.41	76.36 **H**	77.62	71.42	74.39	75.42 **H**
	Top-4	**Imbalanced**	76.42	89.80	82.57	68.61	44.60	54.06	74.73
		Balanced	70.37	75.79	72.98	73.77	68.08	70.81	71.94
KNN	**All**	**Imbalanced**	76.11	88.77	81.95	66.33	44.27	53.10	73.93
		Balanced	66.87	74.71	70.57	71.35	62.99	66.91	68.85
	Top-4	**Imbalanced**	76.14	88.74	81.96	66.34	44.37	53.17	73.95
		Balanced	66.43	73.61	69.74	70.41	62.80	66.39	68.21
Gaussian NB	**All**	**Imbalanced**	82.52	95.19	88.40	41.51	14.49	21.48	79.79
		Balanced	55.73	90.60	69.01	74.88	28.01	40.77	59.31
	Top-4	**Imbalanced**	82.59	95.97	88.78	45.36	14.20	21.63	80.37
		Balanced	56.62	89.59	69.39	75.07	31.35	44.23	60.47

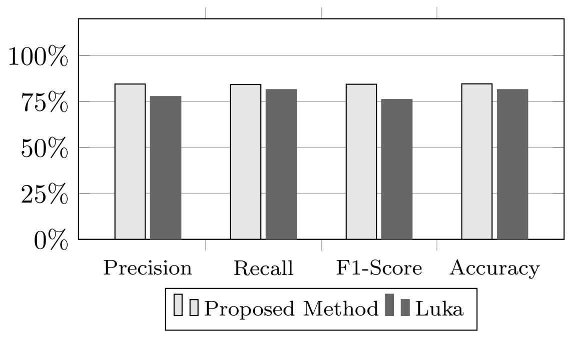
6.6.5 Results

Table 6.10 summarizes the classification results. We apply 10-fold cross validation for training and testing each of the machine learning models. Hence, nine folds of our data are used for training and one fold is applied to testing at any moment and this process iterates until all of the data has been used for training and testing. Then the results are averaged from the 10 models, and the final performance is reported. According to our comparative study, the relationship between question classes and corresponding feature values might be complex. We thus apply the machine learning models (1) on all of the features and (2) on top 4 features. We then analyse the results separately.

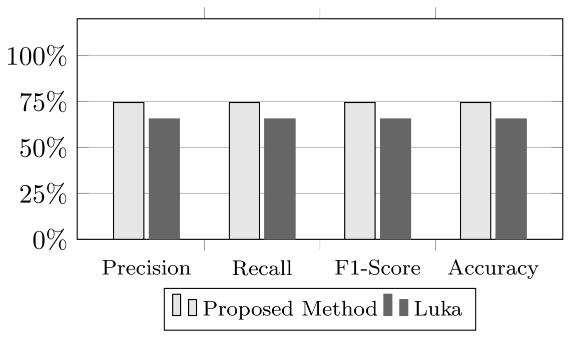
Decision Trees classifies the promoted questions with a maximum of about 91% precision and 90% recall when we use all of the features and an imbalanced dataset. The precision is about 60% for the discouraged questions for imbalanced dataset. The imbalanced dataset might generate suboptimal classification models. When we trained the model on the balanced dataset, the precision increases from about 60% to 75%. The overall classification accuracies for the balanced and imbalanced dataset are about 75% and 85% respectively. When we choose the top-4 features, the evaluation metrics show almost similar results as using all the features. It indicates that the remaining features have very low effect on the classification models. When we train the models with Random Forest and an imbalanced dataset, we see that precision of the discouraged questions increases from about 60% to 66% but recall decreases about 14%. Similar results we got when we train the models with top-4 features. Among the remaining three machine learning models, ANN performs reasonably well when we use a balanced dataset and K-Nearest Neighbors performs well while using top-4 features. K-Nearest Neighbors usually performs better with a lower number of important features and it is not much dependent on the training phase. Both the ANN and Gaussian Naive Bayes perform poor for classifying discouraged questions when we train them with an imbalanced dataset.

As shown in Table 6.10, we mark the highest performance by **H**. We see that when models are trained with all the features, Random Forest and ANN outperform the other models either it is trained with a balanced or an imbalanced dataset. These models have the highest overall accuracy for both the datasets. When we choose the top-4 features to train the model, Decision Trees outperforms the other models with the highest F1-score and overall accuracy for both the questions classes and datasets. This model also have the highest F1-score in classifying discouraged questions with an imbalanced dataset. ANN outperforms the other models with the highest F1-score in classifying promoted questions when we train the model with a balanced dataset. Unfortunately, Gaussian Naive Bayes performs poor in all the settings. Naive Bayes classifier makes a very strong assumption on the shape of the data distribution. That is, any two of the features are either independent or not given the output class. Such an assumption might affect the models performance. Naive Bayes also suffers from imbalanced (resulting in skewed probabilities) classes and continuous features[98].

After analysing the results, it could be concluded that tree-based classifiers (e.g., Decision Trees) can generate a better model to classify the questions based on our dataset and features. Probabilistic models like Naive Bayes could not generate optimal models especially using an imbalanced dataset.

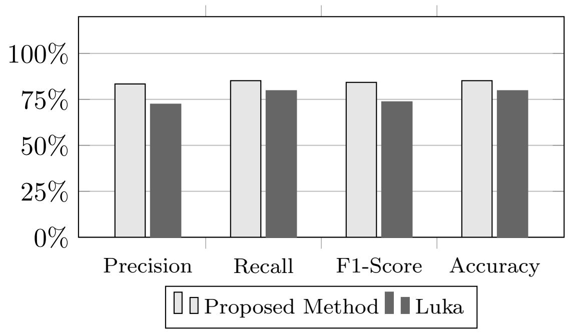


(a) Imbalanced dataset

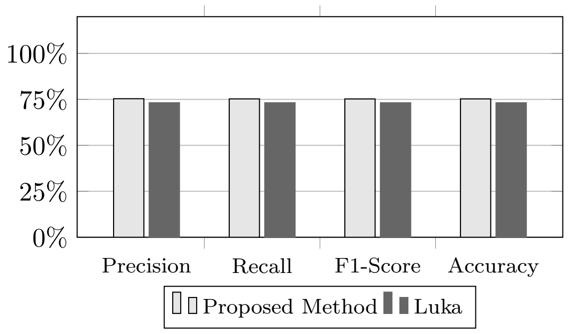


(b) Balanced dataset

Figure 6.14: Results of Decision Trees classifier.



(a) Imbalanced dataset



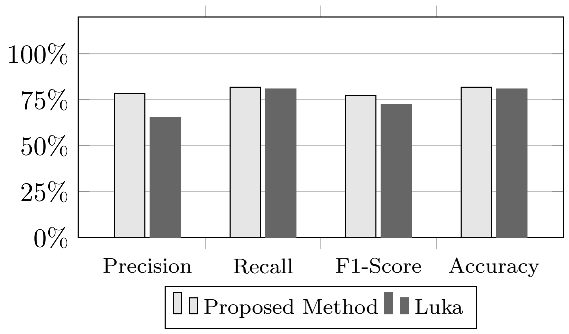
(b) Balanced dataset

Figure 6.15: Results of Random Forest classifier.

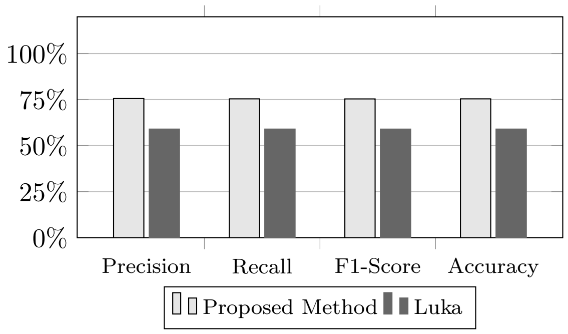
6.6.6 Performance Comparison with Baseline Model

To the best of our knowledge, agreement analysis between subjectivity and objectivity on SO is still an unmet attempt. However, Ponzanelli et al. [93] attempt to classify the questions based on quality and build a machine learning model to discriminate the bad (*i.e., score* <0) and good quality (*i.e., score* >0) questions. They identify three sets of metrics that include textual and non-textual features of SO questions. The metrics are SO (M_{SO}), readability (M_R) and user popularity (M_P). However, the attributes of the user popularity (e.g., up-votes, down-votes) metric are directly or indirectly associated with the subjective evaluation. In this article, we investigate objective quality metrics. We thus select the M_{SO} and M_R metrics to prepare an equivalence set of attributes.

Figures 6.14-6.18 show the comparative results of our models and the baseline models by Ponzanelli et al. [93]. We use the same settings of the machine learning models and identical dataset to compare the models' performances. First, we calculate precision, recall, F1-score, and overall accuracy. Then we compare the values of each of the evaluation metrics between our models and baseline models. When we train our model with decision trees and use an imbalanced dataset, both the F1-score and accuracy are about 85% (Figure 6.14a). They are as low as about 76% and 82%, when we train the model with the attributes used in the

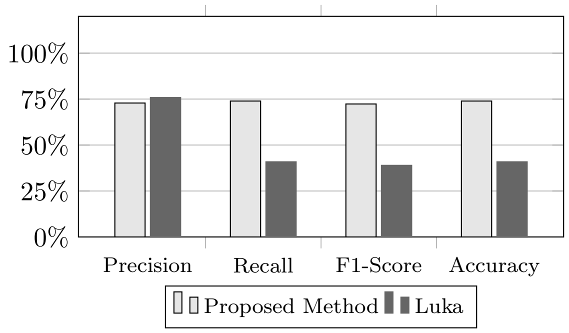


(a) Balanced Imbalanced

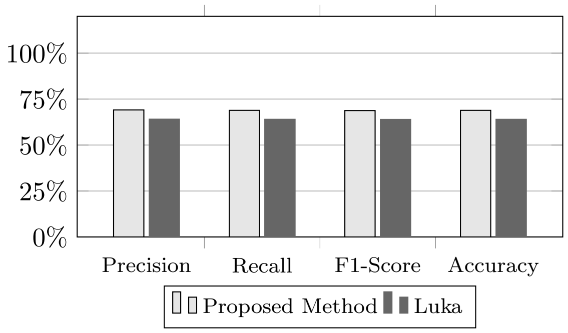


(b) Balanced dataset

Figure 6.16: Results of Artificial Neural Network classifier.



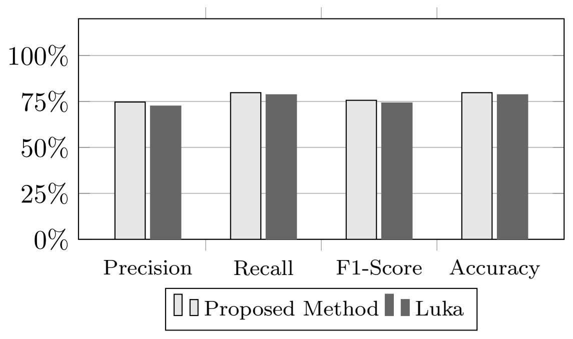
(a) Imbalanced dataset



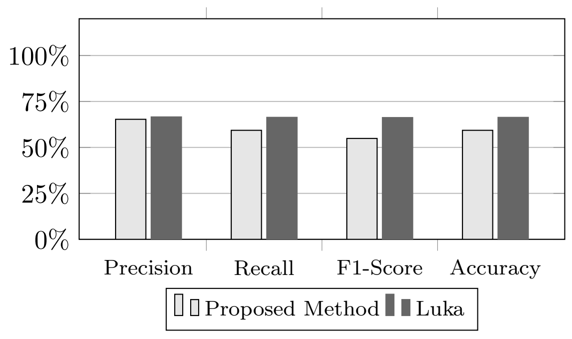
(b) Balanced dataset

Figure 6.17: Results of K-Nearest Neighbors classifier.

baseline study [93]. The performance is also significantly higher when we use a balanced dataset. The F1-score and overall accuracy are about 10% higher than the baseline model (Figure 6.14b). We see that the overall performance of the baseline model improves when we train the model with Random Forest. Nonetheless, our models outperform the baseline models. The F1-score is 10% higher when we use an imbalanced dataset (Figure 6.15a). Our models also outperforms the baseline models when we train them using Artificial Neural Network (Figure 6.16) and K-Nearest Neighbors (Figure 6.17). Here, high dimensionality is a potential factor that reduces the performance of K-Nearest Neighbors. As shown in Figure 6.18b, the model by Ponzanelli et al. [93] outperforms our model only when we train the model with Gaussian Naive Bayes and use a balanced dataset. The underlying data distribution and independent nature of a few of their features might explain the higher performance of their model when using Gaussian Naive Bayes. However, our analysis using five machine learning classifiers with diverse working principles concludes that our features have higher discrimination power to classify promoted and discouraged questions than those of the baseline models.



(a) Imbalanced dataset



(b) Balanced dataset

Figure 6.18: Results of Naive Bayes (Gaussian) Trees classifier.

Table 6.11
Findings summary of Comparative Study

Metric	Promoted	Discouraged	Agreement
Title Quality	–	–	Neither agree nor disagree
Text Readability	Low	High	Disagree
Code Readability	–	–	Neither agree nor disagree
Text-code Ratio	High	Low	Agree
Text-code Correlation	High	Low	Agree
Code Reusability	Low	High	Disagree
Code Understandability	High (Statically typed languages) Low (Dynamically typed languages)	Low (Statically typed languages) High (Dynamically typed languages)	Either agree or disagree
Topic Entropy	Low	High	Agree
Metric Entropy	Low	High	Agree
Sentiment Polarity	–	–	Neither agree nor disagree

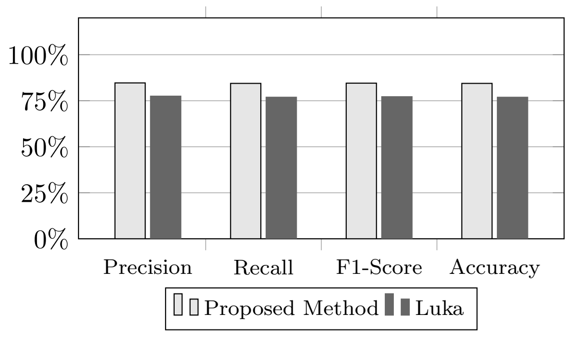
6.6.7 Prediction Model, Performance Evaluation, and Comparison with Baseline Model

We attempt to predict a question will be up voted or down voted during their submission time. Among the five classification models, Decision Trees performs high on average. Thus, we develop a prediction model with Decision Trees. We use the 90% questions that posted in SO earlier to training the model and remaining 10% that posted later to test the model. Although we do not have any time-dependent features, we make sure that we estimate the quality of unseen questions based on past questions.

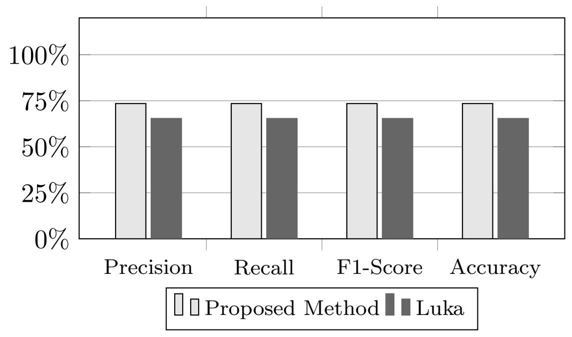
Table 6.12 shows the results of the prediction model. The model predicts the question during their submission time with a maximum of about 87% accuracy. Our model also shows promising results in predicting discouraged question with a maximum of about 76% precision even when the dataset is imbalanced. Figure 6.19 shows the comparative performance between our model and baseline model. We see that our model outperforms the baseline model predicting the questions during the submission time with about 8% accuracy

Table 6.12
Performance of Prediction Model

Classifier	Features	Dataset	Promoted Question			Discouraged Question			Overall
			Precision	Recall	F1-Score	Precision	Recall	F1-Score	Accuracy
DT	All	Imbalanced	90.79	89.87	90.32	58.68	67.21	59.61	84.41
		Balanced	73.57	73.16	73.36	73.34	73.75	73.55	73.46
	Top-4	Imbalanced	91.97	91.15	91.56	63.82	66.65	65.01	86.40
		Balanced	75.24	74.91	75.07	75.06	75.39	75.23	75.15



(a) Imbalanced dataset



(b) Balanced dataset

Figure 6.19: Prediction Results with Decision Trees classifier.

and 7%–8% precision.

6.7 Findings Summary

Our study exposes several insights that might prevent a question from getting downvotes.

Be aware of choosing topics. Questions discussing only common topics (e.g., Python, Java) could prevent the users locate the questions of their interest. Users often discouraged choosing improper topics (e.g., tags) by downvotes. According to our investigation, discouraged questions use more common topics than those of promoted questions.

Code reusability does not always guarantee high quality. Only those code statements should be added that are required to understand reported issues and the redundant code should be avoided. Long and redundant code waste the developers time unnecessarily which might also increase the chance of getting downvotes. In this study, we see that code segments from discouraged questions have a higher parsability rate. However, they have redundant code statements.

Lack of code explanation hurts. A clear explanation of code example might influence the question quality. We also find that the usage of appropriate technical terms is important. A short and not-technical explanation might mislead the users to understand the actual problem.

Sentiment polarity is not a big concern. A few recent studies report that sentiment polarity (e.g., positive, negative) might influence the quality of posts. Calefato et al. [18] find that the sentiment expressed in the answerers' comments might help to estimate answers quality. According to our analysis, sentiment polarity has less impact on the subjective quality assessment of SO questions. In fact, about 50% of the questions have neither positive nor negative sentiment.

Programming language matters. In some cases, the agreement between subjective and objective quality estimation depends on programming languages. For example, code understandability agrees with subjective evaluation for the statically typed programming languages such as Java and C#. However, such quality estimation of the dynamically typed languages such as JavaScript and Python disagrees with subjective evaluation.

6.8 Related Work

Several aspects of SO are investigated in recent studies due to the growing popularity and importance of this Q&A site. Most of the previous studies in mining Q&A sites concentrate on assessing the quality of the posts [7, 21, 93, 94, 101, 144], understanding the way how developers interact among them on Q&A sites [127], providing empirical evidence on writing high-quality questions and answers [13, 20], the usability of the code segments attached in the answers [46, 147], and the impact of the sentiment polarity on getting answers [18, 80].

Ponzanelli et al. [93, 94] devise an approach to classify the questions based on their quality. In particular, they identify the low quality questions to improve the capability of manual review process of SO. They define three sets of metrics to measure the questions' quality namely SO (M_{SO}), readability (M_R) and user popularity (M_P). The attributes of the M_P metric are derived from a direct or indirect evaluation of the users. However, we attempt to assess the quality of questions using objective quality metrics. While Ponzanelli et al. [93] attempted to improve the review process, our goal is to examine either the subjective evaluation of SO questions or answers agree with objective evaluation or not. We measure ten objective quality aspects (e.g., text readability, topic entropy) and build five machine learning models using them to predict promoted and discouraged questions. Our models significantly outperform the models by Ponzanelli et al. [93]. Calefato et al. [21] conduct an empirical assessment to predict the best answers in technical Q&A sites. When they are attempting to assess the quality of answers, we are examining the subjective quality control mechanism of SO with objective quality metrics.

Several studies [24, 35, 35, 39, 68, 106] measure the text readability as a part of objective quality assessment. In this study, we also compute the text readability of the question description to assess the reading ease of description texts. We find a significant difference in text readability between promoted and discouraged questions. Besides text readability, we measure the readability of the code segments included in the SO questions as suggested by many studies [16, 17, 26, 95, 103, 126]. Code segments included in SO are often incomplete and noisy, so the readability is found low for both the question classes. We also measure the understandability of the code segments as existing studies suggest [63, 104, 128]. Zhang et al. [150] report that a large number of APIs hurt the code understandability. We thus count the number of APIs used in the code segment to estimate understandability of code segments.

Several recent studies [46, 147] focus on assessing usability (e.g., parsability) of the code segments of Q&A sites. Yang et al. [147] analyse the usability of 914,974 Java code snippets on SO and report that only 3.89% are parsable. They analyse the code segments found in only the accepted answers of SO. However, we investigation parsability of the code segments included in the question body to estimate their reusability. Although, we found a higher parsing rate than the answer code segments reported by Yang et al. [147], the parsing rate of code segments included in either promoted or discouraged questions is close to each other.

Several studies [18, 20, 110, 135] guide the users to understand the factors for getting fast answers, create quality postings or mining successful answers based on some actionable factors such as information presentation (e.g., title length), ratio of upper-case letters and question posting time. We also find several metrics that affect the questions' quality. For example, users often provide down-votes when the topics of the questions are less focused or ambiguous. Duijn et al. [29] also relate the quality of code segments included in a question to the quality of the question itself. They find that quality questions require quality code. They inspect around 200 Java code segments of SO to find the construct that indicates code quality. Our automatic assessment of the readability of millions of code segments weakly agrees with their findings.

In addition to that several recent works [19, 51, 61, 80, 81] focus on identifying developers' emotions (e.g., positive, negative) by sentiment analysis to the content of communication traces left in collaborative Q&A sites like SO. Novielli et al. [80] claimed that the emotional style of a technical question does influence the chance of promptly getting a satisfying answer. However, we investigate whether the sentiment polarity influences the quality of the SO questions. Hence, the sentiment polarity does not have a noticeable impact on the question quality.

To the best of our knowledge, this is the first attempt to examine the subjective evaluation of SO with objective quality metrics. We conduct a comparative study between subjective and objective evaluation with 2.5 million questions. We also select the key features that might influence the quality of questions. We examine our features' strength using five machine learning models in predicting promoted and discouraged questions that significantly outperforms the baseline models.

6.9 Threats to validity

Threats to *external validity* relate to the generalizability of results. We analyze about 2.5 millions of SO questions. However, the approaches we tried might show results variation when applied to different types of programming problems. To mitigate this issue, we chose to include questions related to four different programming languages - Java, C#, JavaScript and Python in our dataset. Two of them are dynamically typed languages and the remaining two are statically typed languages. An evaluation of our approach that involves questions from four different languages could measure the effect of this threat. The question quality classification models might be sensitive to our dataset. Therefore, we build five different classification models and analyze their performance individually.

Threats to *internal validity* relate to experimental errors and biases [124]. The quality assessment of the SO questions is threatened by the evaluation systems. We assess the quality of questions using existing tools and approaches. However, we manually investigate the questions while the results seem inconsistent. This mixed approach mitigates this threat.

Statistical Conclusion threats concern the fact that whether the data is sufficient to support the claims. We considered statistically significant samples in all our result analyses. We relied on the crowd assessment to collect data, not on the manual inspection of questions that made possible to gather enough data.

Another issue is that the promoted and discouraged question sample are imbalanced. The number of promoted sample is higher than the discouraged sample for all the programming problems. To resolve this imbalanced situation, we undersampled promoted questions to balance the dataset. Then we analyse the results for both the balanced and imbalanced dataset. Thus we mitigate this issue.

6.10 Conclusion

Q&A web sites like SO are providing a new means for programmers to participate in social learning. The quality of SO questions or answers is assessed by users through voting. To keep the evaluation process flawless, we attempt to investigate the agreement between subjective evaluation of questions by the users and objective quality assessment. We use ten objective metrics from the literature to capture all the key aspects of objective quality of a question of the SO Q&A site. We find that topic entropy, metric entropy, text-code ratio, text-code correlation completely agree with the subjective evaluation. Reusability (i.e., parsability) of the code segments and text readability completely disagree with the subjective evaluation. However, code understandability either agree or disagree based on the programming languages, whereas title quality, code readability, and sentiment polarity have the almost equal outcome for both the promoted and discouraged questions. Our investigation might provide an initial benchmark for further agreement analysis between subjectivity and objectivity.

Understanding and classifying question quality is essential to maintain a good user experience of Q&A services. Low quality questions are regarded as less useful for online Q&A sites. We developed five different machine learning models to classify the promoted and discouraged questions. Our machine learning models can classify the questions with a maximum of about 76%-87% accuracy which is significantly higher than that of the state-of-the-art models. We further attempt to predict questions at the time of the question submission. Our machine learning model predicts the discouraged questions automatically with a maximum of about 76% precision. Predicting potentially down-voted questions in advance might greatly help users improve the questions during their submission.

7 Conclusion

7.1 Concluding Remarks

Technical question and answer (Q&A) websites such as SO has emerged as one of the largest repositories of valuable software development knowledge. It accumulates millions of questions related to programming problems and their solutions as answers. The number of questions submitted at SO has been increasing steadily per month. To help the user community, editors and maintenance team of SO, we thus conduct four studies. In particular, we assess the quality of questions (unanswered and answered), code segments included with the questions, editing mechanism, and also the subjective assessment system of SO contrasting it with ten objective quality metrics.

In our first study, we compare unanswered and answered questions of SO quantitatively and qualitatively. We analyze 4.8 million questions and develop machine learning models to predict the unanswered questions during their submission. First, we conduct quantitative analysis and find that topics discussed in a question, the experience of the question submitter, and the readability of question texts are likely to predict whether a question will be answered or not. Second, we manually investigate 200 unanswered and 200 answered questions. According to our investigation, the qualitative findings support the quantitative findings. Our manual analysis also reveals several non-trivial insights, for example, avoid multiple issues in a single question, that could guide users especially novice users to improve their questions. Third, we build four models to predict the unanswered questions during their submission. Our models can predict the unanswered questions with about 78% precision and about 79% overall accuracy, which are significantly higher than that of two state-of-the-art models. We also test our models' performance with questions from two different programming languages (i.e., Go, C++). In this case, our models perform well that ensure the robustness of our models and features.

In the second study, we manually analyze 400 randomly selected questions from SO and investigate the reproducibility of their reported issues using their code segments. We classify the status of reproducibility into two major categories - reproducible and irreproducible. We can reproduce about 68% of issues using the submitted code segments included with the questions after performing minor or major modifications to them. We also expose the challenges of why several issues could not be reproduced using the attached code segments. We then attempt to see the correlation between the reproducibility of issues and answer meta-data. According to our investigation, the reproducibility of question issues is likely to encourage more high-quality responses including the acceptable answers. Thus, the reproducibility of issues can also be treated as a novel metric of question quality for SO which was ignored by the earlier studies.

In the third study, we study how the questions and answers are edited by the SO users. We qualitatively analyze 1,532 edits in SO. We primarily focus on the rollback edits and the ambiguities related to the rollback edits. We find a total of 14 rollback reasons and eight types of ambiguities that are prevalent in the rejected edits. The ambiguities confuse developers while editing. We also develop algorithms to automatically detect that ambiguities that showed a high degree of precision (average 0.99). When we run the algorithms on all rollback edits, we find several ambiguities (e.g., presentation of contents) are consistently present in the edits, while ambiguities such as temporal are rapidly rising (i.e., accidental removal of already approved edits due to a more recent rejected edit).

In our fourth study, we concentrate on the assessment mechanism of SO posts rather than focusing on the content to ensure a non-biased, reliable quality assessment mechanism. In particular, we compare the subjective assessment of questions with their objective assessment using 2.5 million questions and ten text analysis metrics. We find that topic entropy, metric entropy, text-code ratio, text-code correlation completely agree with the subjective evaluation. Reusability (i.e., parsability) of the code segments and text readability completely disagree with the subjective evaluation. However, code understandability either agree or disagree based on the programming languages, whereas title quality, code readability, and sentiment polarity have an almost equal outcome for both the promoted and discouraged questions. We also developed five machine learning models to classify the promoted and discouraged questions. Our machine learning models can classify the questions with a maximum of about 76%-87% accuracy which is significantly higher than that of the state-of-the-art models. We further attempt to predict promoted and discouraged questions during their submission. Our machine learning model predicts the discouraged questions automatically with a maximum of about 76% precision. Discouraged questions hurt the quality of the knowledge base. Thus, predicting potentially down-voted questions in advance might greatly help users improve the questions during their submission.

7.2 Future Works

While in this thesis, we focus on the quality assessment of both the content and assessment mechanism, in the future, we plan to extend the studies and develop tool supports that help the users to ask better questions, attach appropriate code segments, and avoid ambiguous edits. This section discusses our future plans with the research work in the thesis.

Issue Reproducer: In the second study (Chapter 4), we manually investigate 400 questions and attempt to reproduce the issues discussed in the questions using the included code segments. In that study, we only investigate the questions related to Java programming language which is a statically typed programming language. Next, we will investigate the questions related to any dynamically typed programming language (e.g., JavaScript, Python) to see whether we can reproduce the same results. It will confirm the generalizability of our findings. In the future, we also plan to develop an automated tool supports for predicting the reproducibility of a given question issue after analyzing the included code segments with questions.

Ambiguity Protector: In the third study (Chapter 5), We investigate the quality and trust attributes that may arise at the intersection of user collaboration and shared content in developer forums. We consider the body edits of SO and report the rollback reasons and ambiguities of rollback reasons. Next, we will extend this study in two ways – (1) we will consider the tag and title edits, and (2) based on the insights gained, we will focus on the development of a browser plugin that automatically identifies the ambiguities in edits and suggest the users avoid these ambiguities.

Unanswered Question Detector: In the first study (Chapter 3), we build machine learning models to predict the unanswered questions during their submission. In the future, we plan to develop tool supports (e.g., browser plugin) that will detect the potentially unanswered questions in advance during question submission and recommends question submitters to revise and improve their potentially unanswered questions.

References

[1] Choosing the right machine learning algorithm, Accessed on: January 2020. URL `https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f`.

[2] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *Proc. WSDM*, pages 183–194, 2008.

[3] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[4] J. Anderson. Lix and rix: Variations on a little-known readability index. *Journal of Reading*, 26(6): 490–496, 1983.

[5] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider. Answering questions about unanswered questions of stack overflow. In *Proc. MSR*, pages 97–100, 2013.

[6] R. P. Bagozzi and U. M. Dholakia. Open source software user communities: A study of participation in linux user groups. *Journal of Management Science*, 52(7):1099–1115, 2006.

[7] A. Baltadzhieva and G. Chrupała. Predicting the quality of questions on stackoverflow. In *Proc. RANLP*, pages 32–40, 2015.

[8] Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *The annals of statistics*, 29(4):1165–1188, 2001.

[9] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger. Automatically classifying posts into question categories on stack overflow. In *Proc. ICPC*, 2018.

[10] C. M. Bishop. *Pattern recognition and machine learning.* 2006.

[11] B. Boehm and V. R. Basili. Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili*, 426(37), 2005.

[12] S. Boslaugh. *Statistics in a nutshell: A desktop quick reference.* "O'Reilly Media, Inc.", 2012.

[13] 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.

[14] L. Breiman. Random forests. *Machine learning*, 2001.

[15] L. Breiman. *Classification and regression trees.* Routledge, 2017.

[16] R. P. L. Buse and W. R. Weimer. A metric for software readability. In *Proc. ISSTA*, pages 121–130, 2008.

[17] R. P. L. Buse and W. R. Weimer. Learning a metric for code readability. *TSE*, 36(4):546–558, 2010.

[18] F. Calefato, F. Lanubile, M. C. Marasciulo, and N. Novielli. Mining successful answers in stack overflow. In *Proc. MSR*, pages 430–433, 2015.

[19] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. Sentiment polarity detection for software development. *ESE*, 23(3):1352–1382, 2018.

[20] F. Calefato, F. Lanubile, and N. Novielli. How to ask for technical help? evidence-based guidelines for writing questions on stack overflow. *IST*, 94:186–207, 2018.

[21] F. Calefato, F. Lanubile, and N. Novielli. An empirical assessment of best-answer prediction models in technical q&a sites. *ESE*, pages 1–48, 2019.

[22] R. A. Calvo, S. T. O'Rourke, J. Jones, K. Yacef, and P. Reimann. Collaborative writing support tools on the cloud. *TLT*, 41:66–99, 2005.

[23] A. Y. K. Chua and S. Banerjee. Answers or no answers: Studying question answerability in stack overflow. *JIS*, 41(5):720–731, 2015.

[24] M. Coleman and T. L. Liau. A computer readability formula designed for machine scoring. *Journal of Applied Psychology*, 60(2):283, 1975.

[25] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proc. CSCW*, pages 37–46, 2012.

[26] E. Daka, J. Campos, G. Fraser, J. Dorn, and W. Weimer. Modeling readability to improve unit tests. In *Proc. FSE*, pages 107–118, 2015.

[27] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proc. ICML*, pages 233–240, 2006.

[28] Science Direct. Mann–whitney–wilcoxon test, Accessed on: February 2020. URL `https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/mann-whitney-u-test`.

[29] M. Duijn, A. Kucera, and A. Bacchelli. Quality questions need quality code: Classifying code fragments on stack overflow. In *Proc. MSR*, pages 410–413, 2015.

[30] S. Ercan, Q. Stokkink, and A. Bacchelli. Automatic assessments of code explanations: Predicting answering times on stack overflow. In *Proc. MSR*, pages 442–445, 2015.

[31] Stack Exchange. Database schema documentation for the public data dump and sede, 2019. URL `https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede`.

[32] Stack Exchange. StackExchage API, 2019. URL `http://data.stackexchange.com/stackoverflow`.

[33] Stack Exchange. How does reputation work?, Accessed on: December 2019. URL `https://meta.stackexchange.com/questions/7237/how-does-reputation-work`.

[34] M. Fazzini, M. Prammer, M. d'Amorim, and A. Orso. Automatically translating bug reports into test cases for mobile apps. In *Proc. ISSTA*, pages 141–152, 2018.

[35] R. Flesch. A new readability yardstick. *Journal of applied psychology*, 32, 1948.

[36] K. Ganesan. Rouge 2.0: Updated and improved measures for evaluation of summarization tasks. *arXiv preprint arXiv:1803.01937*, 2018.

[37] P. O. Gislason, J. A. Benediktsson, and J. R. Sveinsson. Random forests for land cover classification. *Pattern Recognition Letters*, 2006.

[38] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Proc. NIPS*, pages 513–520, 2005.

[39] R. Gunning. The technique of clear writing. 1952.

[40] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *TSE*, 38(6):1276–1304, 2012.

[41] E. H. S. Han, G. Karypis, and V. Kumar. Text categorization using weight adjusted k-nearest neighbor classification. In *Proc. PAKDD*, pages 53–65, 2001.

[42] D. J. Hand. *Construction and assessment of classification rules.* 1997.

[43] F. M. Harper, D. Raban, S. Rafaeli, and J. A. Konstan. Predictors of answer quality in online q&a sites. In *Proc. CHI*, pages 865–874, 2008.

[44] H. He and E. A. Garcia. Learning from imbalanced data. *TKDE*, (9):1263–1284, 2008.

[45] V. Honsel, S. Herbold, and J. Grabowski. Intuition vs. truth: Evaluation of common myths about stackoverflow posts. In *Proc. MSR*, pages 438–441, 2015.

[46] E. Horton and C. Parnin. Gistable: Evaluating the executability of python code snippets on github. In *Proc. ICSME*, pages 217–227, 2018.

[47] G. Hsieh, R. E. Kraut, and S. E. Hudson. Why pay?: exploring how financial incentives are used for question & answer. In *Proc. CHI*, pages 305–314, 2010.

[48] N. Hudson, P. K. Chilana, X. Guo, J. Day, and E. Liu. Understanding triggers for clarification requests in community-based software help forums. In *Proc. VL/HCC*, pages 189–193, 2015.

[49] W. Hudson. *The Encyclopedia of Human-Computer Interaction*, chapter Card Sorting. The Interaction Design Foundation, 2nd edition, 2013.

[50] M. J. Islam, QM J. Wu, M. Ahmadi, and M. A. Sid-Ahmed. Investigating the performance of naive-bayes classifiers and k-nearest neighbor classifiers. In *Proc. ICCIT*, pages 1541–1546, 2007.

[51] M. R. Islam and M. F. Zibran. Leveraging automated sentiment analysis in software engineering. In *Proc. MSR*, 2017.

[52] S. TK Jan, C. Wang, Q. Zhang, and G. Wang. Towards monetary incentives in social q&a services. *arXiv preprint arXiv:1703.01333*, 2017.

[53] Steve T.K. Jan, Chun Wang, Qing Zhang, and Gang Wang. Towards monetary incentives in social q&a services. Technical report, arXiv preprint arXiv:1703.01333, 2017.

[54] KDnugget. Random forests explained @ONLINE, Access on: January 2020. URL `https://www.kdnuggets.com/2017/10/random-forests-explained.html?fbclid=IwAR15CSJ_pUpe7eJf6Srygn0zfvwGaII5ugjWlXDoKvOEyEKVo80vToXyt_I`.

[55] A. Kittur and R. E. Kraut. Harnessing the wisdom of crowds in wikipedia: quality through coordination. In *Proceedings of the ACM conference on Computer supported cooperative work*, pages 37–46, 2008.

[56] K. R. Lakhani and E. V. Hippel. How open source software works: free user-to-user assistance. *Journal of Research Policy*, 32:923–943, 2003.

[57] K. R. Lakhani and E. Von Hippel. How open source software works: "free" user-to-user assistance. In *Produktentwicklung mit virtuellen Communities.* 2004.

[58] G. Li, H. Zhu, T. Lu, X. Ding, and N. Gu. Is it good to be like wikipedia?: Exploring the trade-offs of introducing collaborative editing model to q&a sites. In *Proc. CSCW*, 2015.

[59] L. Li, D. He, W. Jeng, S. Goodwin, and C. Zhang. Answer quality characteristics and prediction on an academic q&a site: A case study on researchgate. In *Proc. WWW*, pages 1453–1458, 2015.

[60] A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2002.

[61] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proc. ICSE*, 2018.

[62] C. Y. Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[63] J. C. Lin and K. C. Wu. Evaluation of software understandability based on fuzzy matrix. In *Proc. WCCI*, pages 887–892, 2008.

[64] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk. How do api changes trigger stack overflow discussions? a study on the android sdk. In *Proc. ICPC*, 2014.

[65] V. López, A. Fernández, S. García, V. Palade, and F. Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013.

[66] P. B. Lowry, A. M. Curtis, and M. R. Lowry. A taxonomy of collaborative writing to improve empirical research, writing practice, and tool development. *JBC*, 41:66–99, 2005.

[67] P. Luca, B. Gabriele, M. Di Penta, O. Rocco, and L. Michele. Prompter-turning the ide into a self-confident programming assistant. 2016.

[68] G. Harry M. Laughlin. Smog grading-a new readability formula. *Journal of reading*, 12(8):639–646, 1969.

[69] Guillermo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, 10(2): 545–555, 2011.

[70] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge Uni Press, 2009.

[71] S. Marsland. *Machine learning: an algorithmic perspective*. 2014.

[72] M. L. McHugh. The chi-square test of independence. *BM*, 23(2):143–149, 2013.

[73] S. A. McLeod. What does effect size tell you?, 2029. URL `Simplypsychology:https://www.simplypsychology.org/effect-size.html`.

[74] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, pages 775–780, 2006.

[75] S. Mondal, M. M. Rahman, and C. K. Roy. Can issues reported at stack overflow questions be reproduced?: an exploratory study. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 479–489, 2019.

[76] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *Proc. ICST*, pages 33–44, 2016.

[77] L. Moreno, J. J. Treadway, A. Marcus, and W. Shen. On the use of stack traces to improve text retrieval-based bug localization. In *Proc. ICSME*, 2014.

[78] C. Munteanu, R. Baecker, and G. Penn. Collaborative editing for improved usefulness and usability of transcript-enhanced webcasts. In *Proc. SIGCHI*, pages 373–382, 2008.

[79] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example - A study of programming Q and A in Stack Overflow. In *Proc. ICSM*, pages 25–35, 2012.

[80] N. Novielli, F. Calefato, and F. Lanubile. Towards discovering the role of emotions in stack overflow. In *Proc.SSE*, pages 33–36, 2014.

[81] N. Novielli, D. Girardi, and F. Lanubile. A benchmark study on sentiment analysis for software engineering research. *arXiv preprint arXiv:1803.06525*, 2018.

[82] Stack Overflow. Cordova media plugin doesn't work, Accessed on: December 2019. URL `https://stackoverflow.com/questions/38958605/cordova-media-plugin-doesnt-work`.

[83] Stack Overflow. Create tags, Accessed on: December 2019. URL `https://stackoverflow.com/help/privileges/create-tags`.

[84] Stack Overflow. How do i ask a good question?, Accessed on: December 2019. URL `https://stackoverflow.com/help/how-to-ask`.

[85] Stack Overflow. wx.frame error when calling one script from another, Accessed on: December 2019. URL `https://stackoverflow.com/questions/47331054/wx-frame-error-when-calling-one-script-from-another`.

[86] Stack Overflow. What is reputation? how do i earn (and lose) it?, Accessed on: February 2020. URL `https://stackoverflow.com/help/whats-reputation`.

[87] Stack Overflow. Stack Exchage API, Last accessed on: June 2019. URL `https://data.stackexchange.com/stackoverflow/queries`.

[88] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd documentation: Exploring the coverage and dynamics of api discussions on stack overflow. Technical report, Technical Report GIT-CS-12-05, Georgia Tech, 2012.

[89] C. Parnin, C. Treude, L. Grammel, and M. A. Storey. Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. *Georgia Institute of Technology, Tech. Rep*, 11, 2012.

[90] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[91] E. Pitler and A. Nenkova. Revisiting readability: A unified framework for predicting text quality. In *Proc. EMNLP*, 2008.

[92] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3): 21–45, 2006.

[93] L. Ponzanelli, A. Mocci, A. Bacchelli, and M. Lanza. Understanding and classifying the quality of technical forum questions. In *Proc. QSIC*, pages 343–352, 2014.

[94] L. Ponzanelli, A. Mocci, A. Bacchelli, M. Lanza, and D. Fullerton. Improving low quality stack overflow post detection. In *Proc. ICSME*, pages 541–544, 2014.

[95] D. Posnett, A. Hindle, and P. Devanbu. A simpler model of software readability. In *Proc. MSR*, pages 73–82, 2011.

[96] F. J. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proc. ICML*, pages 445–453, 1998.

[97] M. M. Rahman and C. K. Roy. An insight into the unresolved questions at stack overflow. In *Proc. MSR*, pages 426–429, 2015.

[98] J. D. Rennie, L. Shih, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proc. ICML*, pages 616–623, 2003.

[99] M. P. Robillard and R. Deline. A field study of api learning obstacles. *ESE*, 2011.

[100] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *ACM sigmod record*, volume 24, pages 71–79, 1995.

[101] P. K. Roy, Z. Ahmad, J. P. Singh, M. A. A. Alryalat, N. P. Rana, and Y. K. Dwivedi. Finding and ranking high-quality answers in community question answering sites. *Global Journal of Flexible Systems Management*, 19(1):53–68, 2018.

[102] R. K. Saha, A. K. Saha, and D. E. Perry. Toward understanding the causes of unanswered questions in software information sites: a case study of stack overflow. In *Proc. FSE*, pages 663–666, 2013.

[103] S. Scalabrino, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Improving code readability models with textual features. In *Proc. ICPC*, pages 1–10, 2016.

[104] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Automatically assessing code understandability: how far are we? In *Proc. ASE*, pages 417–427, 2017.

[105] J. Skeet. The golden rule: Imagine you're trying to answer the question, 2010. URL `https://codeblog.jonskeet.uk/2010/08/29/writing-the-perfect-question`.

[106] E. A Smith and RJ Senter. Automated readability index. *AMRL-TR. Aerospace Medical Research Laboratories (US)*, pages 1–14, 1967.

[107] M. Soltani, A. Panichella, and A. van Deursen. A guided genetic algorithm for automated crash reproduction. In *Proc. ICSE*, pages 209–220, 2017.

[108] Statistics Solutions. Chi-square test of independence, Accessed on: February 2020. URL `https://www.statisticssolutions.com/non-parametric-analysis-chi-square`.

[109] Statistics Online Support (SOS). Chi-square test of independence, Accessed on: February 2020. URL `http://sites.utexas.edu/sos/guided/inferential/categorical/chi2`.

[110] M. Squire and C. Funkhouser. "a bit of code": How the stack overflow community creates quality postings. In *Proc. HICSS*, pages 1425–1434, 2014.

[111] S. Subramanian, L. Inozemtseva, and R. Holmes. Live api documentation. In *Proc. ICSE*, 2014.

[112] G. M. Sullivan and R. Feinn. Using effect size—or why the p value is not enough. *Journal of graduate medical education*, pages 279–282, 2012.

[113] Stack Overflow Edit System. php curl iis 6.0 windows server 2003, 2008. URL `https://stackoverflow.com/revisions/177112/3`. Online; Last accessed 10 January 2020.

[114] Stack Overflow Edit System. Create an attribute to break the build, 2008. URL `https://stackoverflow.com/revisions/28150/4`. Online; Last accessed 10 January 2020.

[115] Stack Overflow Edit System. How do i get whole and fractional parts from double in jsp/java?, 2009. URL `https://stackoverflow.com/revisions/343584/3`. Online; Last accessed 10 January 2020.

[116] Stack Overflow Edit System. How do i iterate through a string in python?, 2012. URL `https://stackoverflow.com/revisions/228734/3`. Online; Last accessed 10 January 2020.

[117] Stack Overflow Edit System. var functionname = function() vs function functionname() , 2012. URL `https://stackoverflow.com/revisions/336859/5`. Online; Last accessed 10 January 2020.

[118] Stack Overflow Edit System. How do i delete a discipline in epf composer 1.5?, 2013. URL `https://stackoverflow.com/revisions/71332/12`. Online; Last accessed 10 January 2020.

[119] Stack Overflow Edit System. Javascript to extract author/date from svn keyword substitution, 2014. URL `https://stackoverflow.com/revisions/116690/5`. Online; Last accessed 10 January 2020.

[120] Stack Overflow Edit System. What soap client libraries exist for python, and where is the documentation for them?, 2016. URL `https://stackoverflow.com/revisions/206964/15`. Online; Last accessed 10 January 2020.

[121] Stack Overflow Edit System. What should main() return in c and c++?, 2018. URL `https://stackoverflow.com/revisions/204483/7`. Online; Last accessed 10 January 2020.

[122] Stack Overflow Edit System. Convert a class to a function when there's __construct() elements, 2019. URL `https://stackoverflow.com/revisions/13221863/5`. Online; Last accessed 10 January 2020.

[123] J. A. Thomas and T.M. Cover. Elements of information theory. *Wiley- Interscience*, pages 187–202, 1991.

[124] Y. Tian, D. Lo, and J. Lawall. Automated construction of a software-specific word similarity database. In *Proc. CSMR-WCRE*, pages 44–53, 2014.

[125] Stat Trek. Chi-square test of independence, Accessed on: February 2020. URL `https://stattrek.com/chi-square-test/independence.aspx`.

[126] C. Treude and M. P. Robillard. Understanding stack overflow code fragments. In *Proc. ICSME*, pages 509–513, 2017.

[127] C. Treude, O. Barzilay, and M.A. Storey. How do programmers ask and answer questions on the web? (NIER Track). In *Proc. ICSE*, pages 804–807, 2011.

[128] A. Trockman, K. Cates, M. Mozina, T. Nguyen, C. Kästner, and B. Vasilescu. "automatically assessing code understandability" reanalyzed: Combined metrics matter. pages 314–318, 2018.

[129] A. Trockman, K. Cates, M. Mozina, T. Nguyen, C. Kästner, and B. Vasilescu. Automatically assessing code understandability reanalyzed: combined metrics matter. In *Proc. MSR*, pages 314–318, 2018.

[130] G. Uddin and F. Khomh. Automatic summarization of API reviews. In *Proc. ASE*, page 12, 2017.

[131] G. Uddin and F. Khomh. Automatic opinion mining from API reviews from stack overflow. *TSE*, 2019.

[132] G. Uddin and M. P. Robillard. How api documentation fails. *IEEE Software*, 2015.

[133] G. Uddin, O. Baysal, L. Guerrouj, and F. Khomh. Understanding how and why developers seek and analyze API-related opinions. *TSE*, pages 1–40, 2019.

[134] S. Wang, T. H. P. Chen, and A. E. Hassan. How do users revise answers on technical q&a websites? a case study on stack overflow. *TSE*, 2018.

[135] S. Wang, T.H. P. Chen, and A. E. Hassan. Understanding the factors for fast answers in technical q&a websites. *ESE*, 23(3):1552–1593, 2018.

[136] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[137] M. White, M. Linares-Vásquez, P. Johnson, C. Bernal-Cárdenas, and D. Poshyvanyk. Generating reproducible and replayable bug reports from android application crashes. In *Proc. ICPC*, pages 48–59, 2015.

[138] Wikipedia. Readability, Accessed on: April 2018. URL `https://en.wikipedia.org/wiki/Readability`.

[139] Wikipedia. Machine learning, Accessed on: February 2020. URL `https://en.wikipedia.org/wiki/Machine_learning`.

[140] Wikipedia. Mann–whitney–wilcoxon test, Accessed on: February 2020. URL `https://en.wikipedia.org/wiki/Mann-Whitney\_U\_test`.

[141] Wikipedia. Effect size, Accessed on: February 2020. URL `https://en.wikipedia.org/wiki/Effect\_size\#Cohen's\_d`.

[142] Wikipedia. Type i and type ii errors, Accessed on: February 2020. URL `https://en.wikipedia.org/wiki/Type_I_and_type_II_errors`.

[143] Wikipedia. Stack Overflow, Accessed on: January 2020. URL `http://en.wikipedia.org/wiki/Stack_Overflow`.

[144] X. Xia, D. Lo, D. Correa, A. Sureka, and E. Shihab. It takes two to tango: Deleted stack overflow question prediction with text and meta features. In *COMPSAC*, volume 1, pages 73–82, 2016.

[145] Y. Ya, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu. Want a good answer? ask a good question first! Technical report, arXiv preprint arXiv:1311.6876, 2013.

[146] Y. Ya, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu. Detecting high-quality posts in community question answering sites. *Journal of Information Sciences*, 302:70–82, 2015.

[147] D. Yang, A. Hussain, and C. V. Lopes. From query to usable code: an analysis of stack overflow code snippets. In *Proc. MSR*, pages 391–402, 2016.

[148] C. Zaiontz. Real statistics using excel, Accessed on: February 2020. URL `https://www.real-statistics.com/non-parametric-tests/mann-whitney-test/cliffs-delta/`.

[149] H. Zhang, S. Wang, T. H. Chen, and A. E. Hassan. Reading answers on stack overflow: Not enough! *TSE*, 2019.

[150] H. Zhang, S. Wang, TH P. Chen, Y. Zou, and A. E. Hassan. An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering*, 2019.

[151] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim. Are code examples on an online q&a forum reliable? a study of api misuse on stack overflow. In *Proc. ICSE*, page 12, 2018.