Please cite as:
Squire, M. and Funkhouser, C. (2014).
"A Bit of Code": How the Stack Overflow Community Creates Quality Postings.
In Proceedings of the 47th Hawai'i International Conference on System Sciences (HICSS47).
IEEE. Big Island, HI, USA. 1425-1434.
DOI: 10.1109/HICSS.2014.185

"A bit of code": How the Stack Overflow Community Creates Quality Postings

Megan Squire
Elon University
msquire@elon.edu

Christian Funkhouser
christian.funkhouser@gmail.com

Abstract

The Stack Overflow web site is an online community where programmers can ask and answer one another's questions, earning points and badges. The site offers guidance in the form of a Frequently Asked Questions (FAQ), beginning with "What kind of questions can I ask here?" The answer explains that "the best Stack Overflow questions have a bit of source code in them". This paper explores the role of source code and non-source code text on Stack Overflow in both questions and answers. The primary contribution of this paper is to provide a more detailed understanding of whether the presence of source code (and how much) actually will produce the "best" Stack Overflow questions or answers. A second contribution of this paper is to determine how the non-code portions of the text might also contribute the "best" Stack Overflow postings.

1. Introduction

Stack Overflow [1] is a web site for programmers to ask and answer technical questions. The guidance for prospective authors on the Frequently Asked Questions page in answer to the question "What kind of questions can I ask here?" is as follows:

> *Stack Overflow is for professional and enthusiast programmers, people who write code because they love it. We feel the best Stack Overflow questions have a bit of source code in them…*[2]

The purpose of this portion of the FAQ is to encourage the authors to write the best possible questions, so as to receive the most satisfying answers that will help themselves and others facing the same technical problem. This is a social community created around high quality, highly technical Q-and-A. One of the main artifacts of the programming process is source code, but the assumption in this FAQ is that most questions and answers will also include sentences written (in the English language [3]) to accompany and clarify the code. Some guidance is therefore given in the FAQ for how much of that source code to include.

Our paper examines, in detail, two of the important phrases used in that FAQ: "best" as it refers to the quality of questions, and "bit of" as it refers to an amount of source code that should be present in questions in order to make them effective. What might these terms "best" and "a bit of" mean in practice? How can contributors to this site understand the advice given in the FAQ, so as to write the "best" Stack Overflow questions and answers to help the community of users?

To answer these questions, this paper presents an analysis of a publicly-available data set containing all content from the Stack Overflow web site. This data set includes all posted questions, answers, scores, and most (non-private) user information. We clean and store the data, then we apply a range of analyses and visualization techniques, in order to shed light on the way the community uses text to produce quality postings.

Section 2 gives relevant background information for how the Stack Overflow community works, including terminology and its mechanisms for encouraging quality in postings. Section 3 enumerates our research questions. Section 4 examines the data set we used to answer these research questions. Section 5 discusses the analyses and findings for our research questions. Section 6 gives some ideas for future work. Section 7 summarizes our conclusions.

2. Stack Overflow Background

Stack Overflow was created in 2008 by software developers Jeff Atwood and Joel Spolsky. As of this writing, the site has 2.3 million registered users who have asked each other more than 5.6 million questions. There are now more than 100 topic-specific sites in the Stack Exchange network. (Stack Overflow is the only one of these sites we will be discussing in this paper.) Alexa.com shows that StackOverflow.com is in the top 80 most visited web sites in the world (top 65 in the USA and top 20 in India). Within the software development community, it is an extremely popular site.

Stack Overflow uses a particular vocabulary to describe its artifacts that users of other social media tools will recognize. Some terms are highlighted here.

Posts: A *post* or *posting* is either a question, or an answer to a question. Posts are given a unique ID number on the Stack Overflow site, and each post has an identifier for its type (for example, type 1 is a question, type 2 is an answer).

Users: A *user* is a registered member of the Stack Overflow web site. In terms of the analysis in this paper, users are participants in the Stack Overflow community. They write posts, write comments, vote on posts, edit posts, etc. In the Stack Overflow community, users do not "friend" each other or "follow" each other or the like. The main interaction between users is to ask or answer each other's questions, or to comment on a posting written by another user.

Reputation: A user builds up reputation points by posing questions, issuing answers, and editing posts. Different actions on the site require different levels of reputation points. As the FAQ explains, "Reputation is a rough measurement of how much the community trusts you." Reputation is an important aspect of the Stack Overflow site, as users compete to build large reputation scores and gain privileges.

Votes: A user with enough reputation points can extend a *vote* in favor of a posting (called an upvote) or against a posting (called a downvote). Votes reflect the user's opinion of the usefulness of a question or the merit of an answer. Voting is a privileged action; in order to begin voting on posts, a user must have built up enough reputation to earn this privilege.

Scores: A score is calculated for a post based on a number of upvotes (+1) and downvotes (-1) given by users who vote on a post. Posts (both questions and answers) with high scores yield reputation points to their authors. Posts with low scores will actually lower the author's reputation points. There is an evolving formula determining how scores affect reputation [4].

Accepted answer: A user who poses a question to the site has the ability to choose one of the submitted answers to be the accepted answer. This presumably reflects that this is the "best" answer, as judged by the original asker.

Favorite: A user can select a question (but not an answer or comment) as a "favorite". This is sort of like bookmarking or saving the question for later. No reputation is conferred when a question is selected as a favorite.

Badges: A user can earn named badges for completing particular actions on the site. These badges appear on the user profile, and a count of different types of badges appear next to the user's name in all postings. An example of a badge is "Copy Editor" (editing more than 500 questions on the site, a gold level badge) or "Yearling" (Active member for a year, earning at least 200 reputation, a silver level badge). There are more than 75 different badges available as of this writing, in gold, silver, and bronze categories. [5]

3. Research Questions

We have organized our work around 5 research questions, all designed to tease out what "best" and "a bit of source code" mean in the Stack Overflow FAQ. The goal is to provide an additional level of detail to users of this site for how to ask and answer high-quality questions.

Q1. Do higher scoring questions include more source code? What about higher scoring answers?

Q2. Do questions with high favorite counts have more source code?

Q3. Do answers chosen as "accepted answers" have more source code than answers not chosen?

The following questions are designed to try to understand what other factors in the non-code portion of the text may make a posting popular (where popularity results in a high score or a high favorite count).

Q4. Do the "best" questions (those with high scores or high favorite counts) also have high scores for readability?

Q5. Do the "best" questions (those with high scores or high favorite counts) also have high scores for sentiment or mood?

4. Data Set

The data collected for this project was the August 2012 XML dump provided by Stack Overflow on the web site. The data was read into a MySQL database for ease of processing. (Stack Overflow releases a dump like this every month.)

The data dump provided includes all the questions, answers, comments, tags, scores, and much (but not all) of the user information. In addition, the data dump comes with pre-computed counts for favorites, upvotes, and downvotes. Figure 1 (next page) shows the data model for the six tables most relevant to this paper.

4.1. Tables

The main table we used for this study was the *posts* table, which includes the following attributes:

- id: a unique number for a posting;

- post_type_id: one of seven types, concentrating especially on 1 and 2 (1=question and 2=answer);
- accepted_answer_id: the post "id" chosen by the original asker as the best (this column uses the "id" column as a foreign key);
- score: the number of upvotes and downvotes for a post;
- body: the content of the posting (including code, text, images, links);
- answer_count: how many answers did this question get?
- favorite_count: how many users selected this question as a favorite?

The *new_posts_meta* table was created by us for use on this project. It has a one-to-one relationship with the *posts* table, since we did not want to modify the structure or data of the original *posts* table, but we wanted to add extra attributes for each record in the *posts* table. We then wrote several scripts to process the "body" column in the *posts* table, stripping out different levels of code and calculating various scores, as described below. This new table includes:

- The posts.body column as plaintext only (html tags stripped out);
- The posts.body column as plaintext with source code stripped out;
- The code-text ratio of the posts.body column text (source code versus non-source code);
- Readability scores of the plaintext (Flesch-Kincaid Grade Level and Flesch-Kincaid Reading Ease scores [6]) for plaintext without any source code or markup;
- Sentiment scores [10] for plaintext without source code or markup.

We describe more about readability and sentiment scoring in Section 5.4 and 5.5.

4.2. Metrics

Some basic metrics about the Stack Overflow data set are as follows:

- The data set is about 80GB.
- There are approximately 10.3 million rows in each of the *posts* table and the *new_posts_meta* table.
- The votes table has about 27 million rows in it.
- The comments table has about 14 million rows in it.
- The users table has about 1.1 million rows.
- The post_history table (users making posts) has about 26.2 million rows.
- The range of (non-administrative) user post counts goes from 0 (never posted) to one user who has made 43,700 posts.

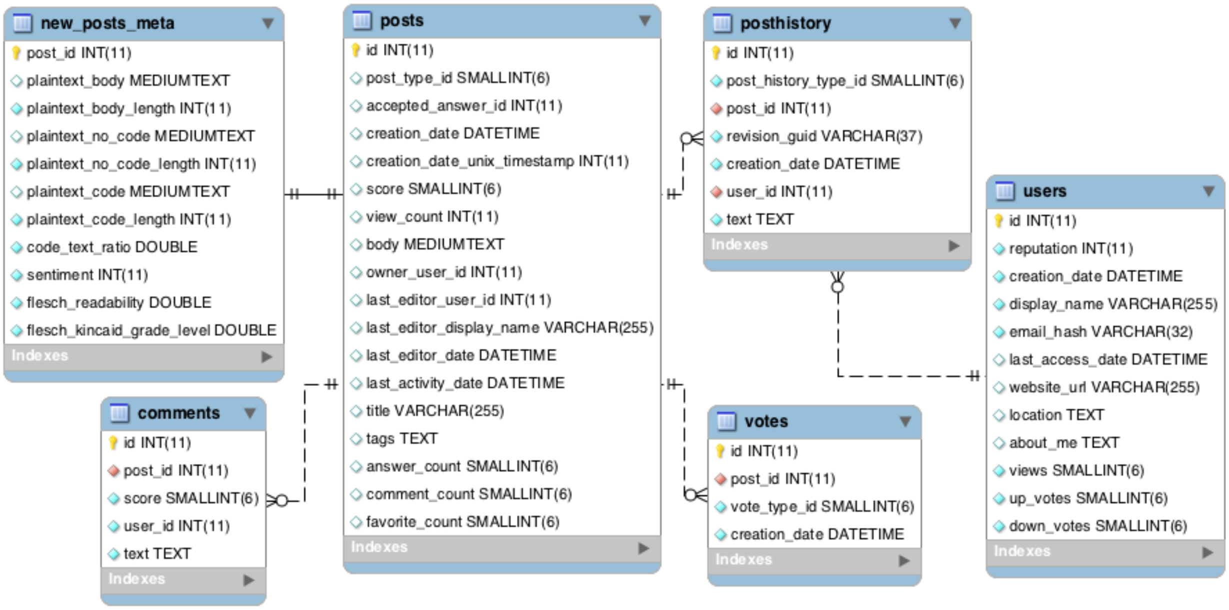


Figure 1. Stack Overflow Data Model

5. Analysis

In this section we describe our attempts to answer each of our research questions described in Section 3, using the data set from Section 4. In each case, we visualize the data to help guide our conclusions.

5.1. Findings for Q1

The first research question was “Do higher scoring questions have more source code? What about higher scoring answers?” To answer this, we used our new_posts_meta table with the plaintext_nocode_length and plaintext_code_length columns. We then calculated a code-text-ratio for each posting as a percentage of the total post that is code. (Values of 0 are where a posting was all text and no code, whereas a value of 1 is where the posting was all code and no text).

For this analysis, we first determined that 95% of question scores fall into a range of between -1 and 8, whereas scores for answers are between 0 and 9. (Lower scores are worse.) We then determined that 95% of all questions had a code-text-ratio of between 0 and .91, and answers had a code-text-ratio between 0 and .98. We then used these ranges to select the reasonable set of scores (those that were like 95% of cases) and code-text-ratios for questions and for answers. Finally we visualized the ratios using boxplots in Figure 2. The plots are shown with one box per score (-1 through 8 for questions, 0 through 9 for answers).

With questions, as scores increase left to right on the x-axis, the code-text-ratios decrease (with the exception of the first jump from -1 to 0 score).

We can contrast this to what happens with answers (on the right, in Figure 2). With answers, as we move from left to right on x-axis, the higher scores also have a higher code-text-ratio.

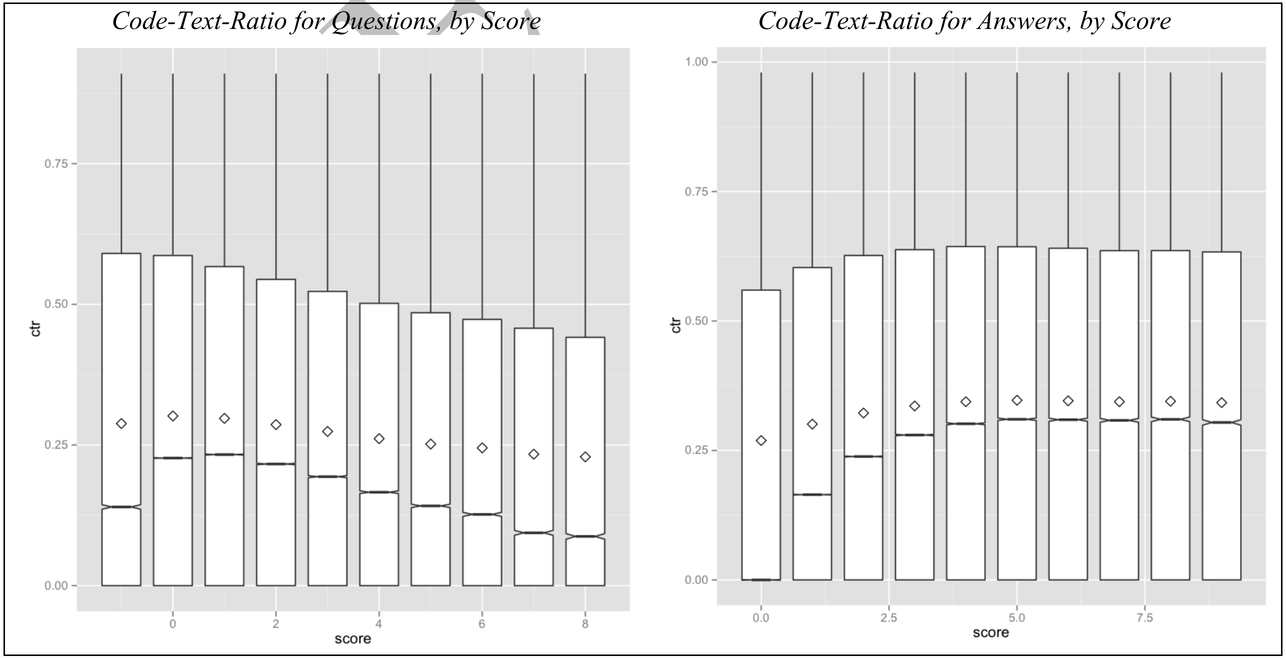
For *answers*, the ideal code-text-ratio seems to be about 1:3. The highest scoring (typical) answers will have one line of code for every 3 lines of non-code text. (Or, 25% of the total post length is code.)

However, *questions* with large amounts of code are scored lower. We show that the highest scoring questions had a ratio of about 1:9, or one line of code for every 9 lines of non-code text.

“Look at my code and tell me what is wrong with it” types of questions may be considered less interesting or less developed, and thus downvoted, whereas answers are considered better when they have more source code.

5.2. Findings for Q2

Our second research question was “Do questions with higher favorite counts have more source code?” Recall that favorites do not affect a user’s reputation, but they could be considered a measure of social worth of a question since people use the “favorite” as a way of “bookmarking” the question for later perusal. Answers do not have the ability to be favorites.


Code-Text-Ratio for Questions, by Score

Code-Text-Ratio for Answers, by Score

Figure 2. Q1: Code-Text-Ratios (0-1) for Questions and Answers, with Scores (-1-8)

To answer Q2, we simply selected questions from the *posts* table, (post_type_id = 1) and the favorite count for each post, as given in the original data dump. To ensure that we got really popular questions for this analysis of "best" questions, we selected only posts with a favorite count above 10. (More than 10 favorites would be considered an extremely popular question, since 95% of Stack Overflow posts have a favorite count of 10 or less.)

Figure 3 shows a simple scatter plot with code-text ratios on x-axis and favorite counts on y-axis (log scale).

Among these very "best" questions, Figure 3 shows that it is slightly more unusual to have a high favorite count *and* a very high code-text ratio (above .8 or so). Thus, these results are very similar to the results for Q1 with questions.

It is common for the highest "favorites" to have a low code-text ratio. There is also a number of posts with a code-text ratio of 0 that have a high favorite count. These tend to be lists of things, such as books about programming or web sites on a certain programming topic.

5.3. Findings for Q3

Our third question was "Do accepted answers have more source code than non-accepted answers?" For this analysis, we again selected the code-text ratios for two groups: the answers that were accepted as "best" by the original authors, and the rest of the answers. Figure 4 (next page) shows the density distributions of the answers in each of these two groups.

As Figure 4 shows, it is most probable that both accepted and non-accepted answers have low code-text ratios, thus the tall spike on the left. However, among those answers that *do* include code (moving away from 0 on the x-axis) it is *less* likely (density is more negative) that non-"accepted" answers will have a high code-text ratio, and *more* likely that "accepted" answers will have a high code-text ratio. This agrees with findings from Q1 and Q2.

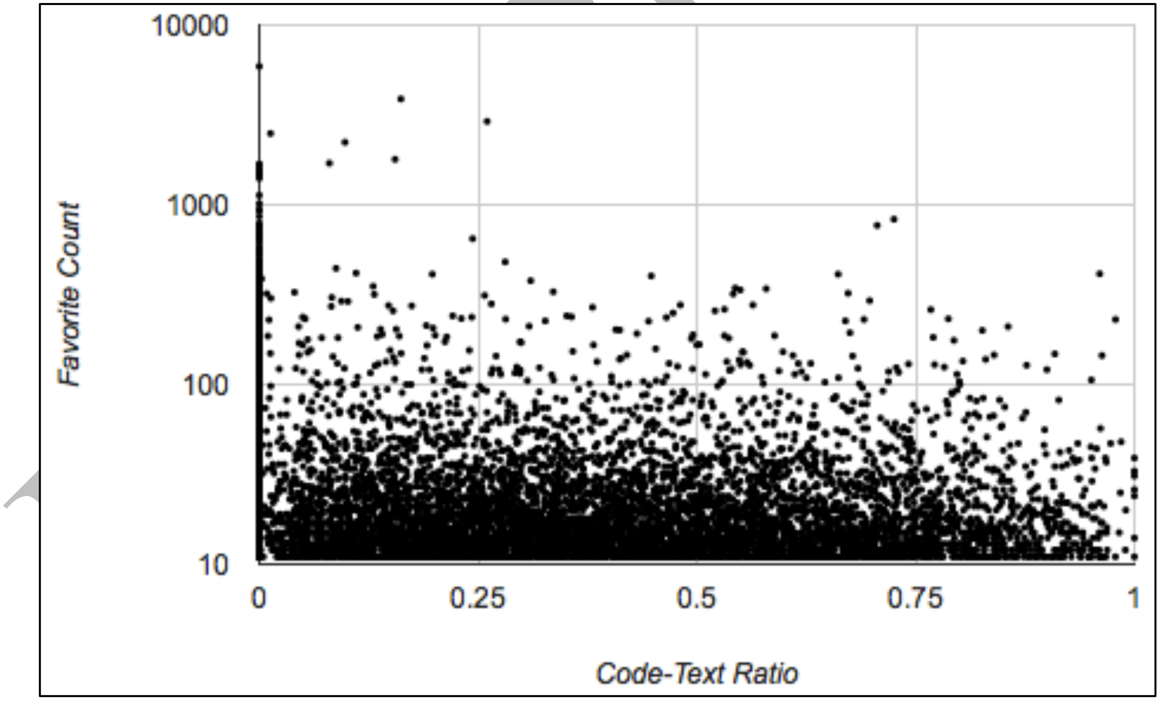


Figure 3. Q2: Code-Text-Ratios (0-1) for Questions and Answers, with Scores (-1-8)

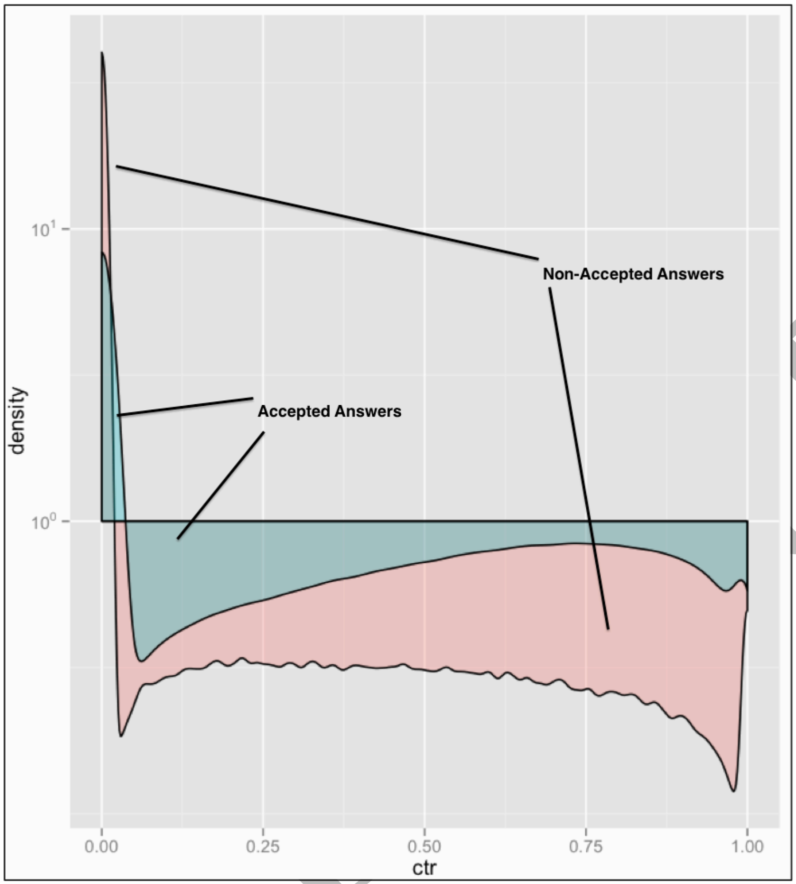


Figure 4. Q3: Code-Text-Ratios for Accepted and Non-Accepted Answers

From these analyses in sections 5.1-5.3, we determine that the presence of source code plays some role in how a user will score a question and an answer. In general it appears that users like to see a smaller amount of code in questions, and a larger amount in answers.

Following this, we became curious what other textual factors could also result in the "best" questions or answers. We were especially interested in factors that could be affected by code-text ratio. In other words, what are authors doing with the non-code portion of their posting? Is the non-code portion of a good post more readable? Is the sentiment of a good post particularly positive or upbeat? Is humor worth anything in getting a good score? Q4 and Q5 attempt to illuminate this aspect of formulating the "best" postings on Stack Overflow.

5.4. Findings for Q4

The Flesch-Kincaid Reading Ease (FKRE) [6] metric is a simple measure of the alleged readability of a piece of text. (As a metric, it has some flaws, which are outlined in [7].) It is calculated using the following formula, for example as shown in [8]:

$$206.835 - 1.015\left(\frac{wordCount}{sentenceCount}\right) - 84.6\left(\frac{syllableCount}{wordCount}\right)$$

In the FKRE results, lower scores are more difficult to read, and higher scores equate to more ease in reading the document. Scores of less than 30 are an indication of university-level reading. Scores of 90-100 are used to indicate readings appropriate to a 10-year old child.

We used this formula on the plaintext_no_code column in our *new_posts_meta* table, and saved the results as a new column in the *new_posts_meta* table.

The following simple, one-line Stack Overflow answer scored a 108.2 on the FKRE:

```
Could you make it cookie-based?
```

The next question is an example of text that scored a middle-of-the-road 45.75 on the FKRE:

```
What's the simplest way in Ruby to group an
array of arrays by element order? In other
words, to get all the 0th elements, then all
the 1th elements, etc.
```

The following answer to a question garnered a very low FKRE score of 5.1:

```
I used NSGA-II in a multi-objective
evolutionary approach to optimize an
artificial neural network that corresponds to
a computational model of a part of the brain
which is supposed to be a low-level system for
action selection. If you are interested you
may find more information on

http://francky.me/publications.php#mRF2011
Note that any other Pareto-compliant ranking
method would have probably worked.
```

The biggest difference between these postings is the number of words per sentence and the number of syllables per word.

Figure 5 shows the "favorite post" counts plotted against readability for the top 1000 most favorite questions. The easier-to-read questions will approach 100. Harder-to-read questions may even have a negative score. Figure 6 shows scores plotted against FKRE for the top 1000 highest scoring posts.

In both figures, there is a tight cluster of expected readability values for most of the postings (between 30-80), and even the highest scoring/most favorite postings fall into this same readability range.

For those top 1000 favorite posts, the range of favorite counts is 5894 at the high end and 71 at the low end. For the top 1000 scoring posts, the range of scores is 2499 at the high end to 126 at the low end.

After seeing this, we became interested in the very-low-readability (difficult) postings that still get high favorite counts or high scores. What does an unreadable-but-still-great post look like? Can finding these odd postings help us find out whether there is something wrong with the readability metric?

Upon manually inspecting these unreadable-but-still-great posts, it appears that most are of these main types:

(1) postings that are long lists of communally-edited resources, such as lists of books or web sites, and

(2) postings that are quite long and cover topics with odd syntax, such as design of programming languages, the use of regular expressions, or methods for text analysis, and

(3) postings that include long lists of filenames or URLs, or error messages pasted in as text.

For (1), the titles of books or blogs will rarely be complete sentences, thus lowering the denominator in the first part of the FKRE equation. For (2) and (3), posts on these topics tend to have partial words, word stems, and sequences of punctuation and characters that are not considered "readable" (yet they are also not "source code" either). This can explain the low FKRE scores for these popular postings since the simple FKRE algorithm could become confused.

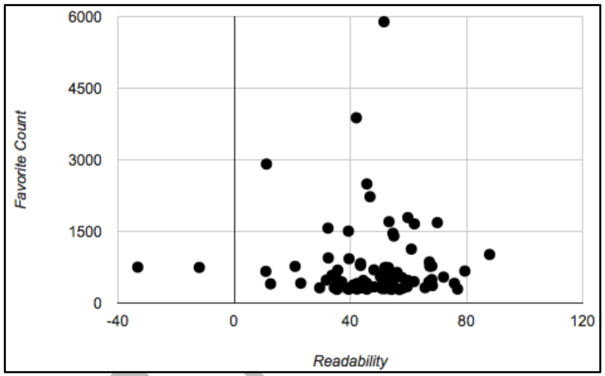


Figure 5. Q4: Readability scores for the top 1000 most favorite posts

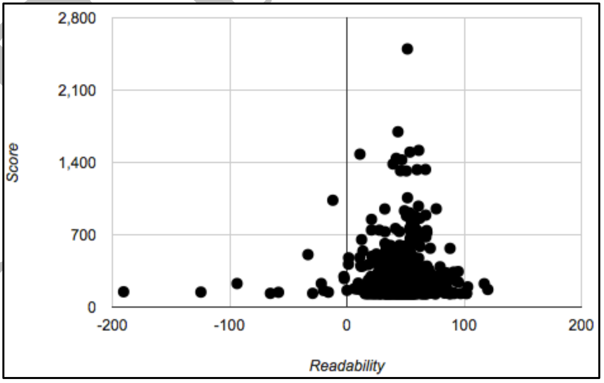


Figure 6. Q4: Readability scores for the top 1000 highest scoring posts

5.5. Findings for Q5

The purpose of Q5 is to discern whether "best" questions (those with high scores or high favorite counts) also have high scores for sentiment. To calculate sentiment, we used a popular lexicon [9] described in [10] that organizes words into positive and negative lists. We calculated the sentiment of a plaintext / no-code body of text with a simple system (similar to the way Stack Overflow calculates its own scores for posts): +1 for positive words and -1 for negative words. Zero is a neutral score. More positive postings will have positive integers. Negative sentiment postings will have a negative score.

Below are three examples of Stack Overflow questions and their associated sentiment scores. Each of these postings appeared on the "top 1000" list of favorite postings.

Here is an example of a question that received a score of 0 for sentiment (includes zero positive or negative words):

```
Task: Print numbers from 1 to 1000 without
using any loop or conditional statements.
Don't just write the printf() or cout
statement 1000 times. How would you do that
using C or C++?
```

Below is an example of a question that scored a 3 for sentiment (includes three positive words: like, free, hope; includes zero negative words):

```
I would like to know how I can find the memory
used on my Android application,
programmatically.

I hope there is a way to do it. Plus I would
like to understand how to get the free memory
of the phone too.
```

Finally, here is an example of a posting with a score of -4 for sentiment (includes six negative words: strangest, forced, strangest (again), worst, odd, ill; includes two positive words: please, funniest):

```
What was the strangest coding standard rule
that you were ever forced to follow? And by
strangest I mean funniest, or worst, or just
plain odd. In each answer, please mention
which language, what your team size was, and
which ill effects it caused you and your team.
```

Figure 7 shows the favorite counts plotted against sentiment for the top 1000 most often "favorite" questions. Figure 8 shows scores plotted against sentiment for the top 1000 highest scoring posts.

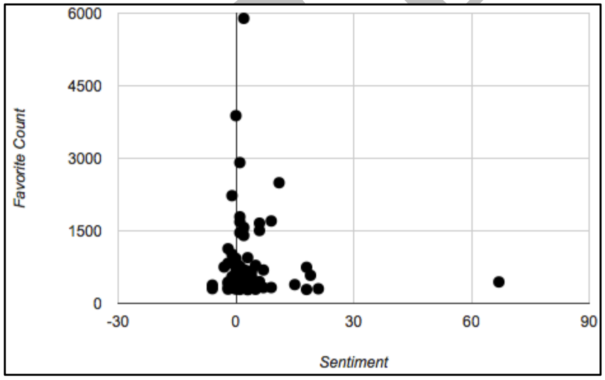


Figure 7. Q5: Sentiment scores for the top 1000 most favorite posts

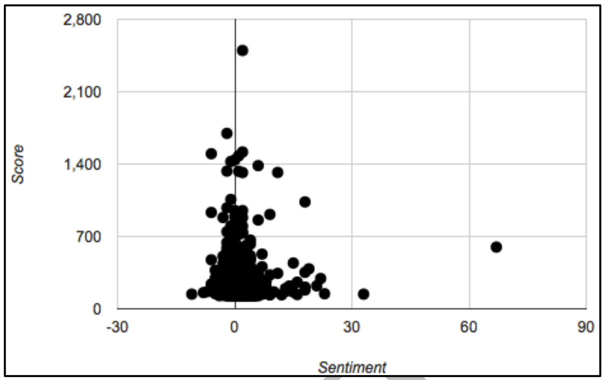


Figure 8. Q5: Sentiment scores for the top 1000 highest scoring posts

In both figures, there is a tight cluster of postings with neutral sentiment values (near 0). We observe that even the highest scoring/highest favorite postings fall into this same sentiment range near 0.

However, we also observe a few postings with very high sentiment values. Upon manually inspecting these posts, it appears that the very high sentiment postings are usually relatively long and they describe one of the following items: (a) contests or challenges that the original author is trying to get people to participate in, (b) lists of recommended resources or books that have been well-reviewed for a particular topic, or (c) summaries of effective programming techniques. Some of the non-code-related postings (for example contests and lists of books) have since been closed for new comments/scores/answers by site moderators for not following the intent of the site, but these posts are still able to keep their high scores that they earned while they were still active.

In any case, our method of sentiment scoring is easily affected by the length of the original posting, so high sentiment posts will have a lot of happy/positive words for their size. Examples include postings enticing people to enter a programming contest that is "fun", "enjoyable", "great"; or postings describing an algorithm as "new", "easy", "quick", and "fast"; or those describing a list of tutorials that are "brilliant", "complete", and "excellent".

Some popular questions are negative, but the negative questions are much less dramatic in their sentiment scores. For instance, -11 was the lowest sentiment score we saw for a top-1000 posting, but 67 was the highest. The average was 1.215. Upon inspection, it seems that the negativity in these posts comes from the original author bemoaning his or her status as confused about some problem for which they were seeking a solution, often for humorous effect, exaggerating the story of what happened to them during a programming experience. For a good example of this, see [10], a humorous posting that has a very high score of 1775 and 340 favorites, but a very low sentiment score of -6. (It also happens to have a FKRE in the middle of the pack, at 53.7.)

6. Limitations of the work

We have alluded to several shortcomings with the FKRE and sentiment scoring algorithms. In this section, we will summarize the limitations for *each* piece of the project in turn. This includes identifying issues with the data set itself, and with the methods of analysis.

6.1. Limitations relating to the data set itself

Age. As described earlier in Section 3, the data set we used for this paper was a snapshot created in August 2012. The most obvious next step is to update the analyses using a more recent data set. Next, we would like to add some analyses that take into account changes over time. Stack Overflow has existed since 2008, but has the use of language and especially source code in questions and answers changed over time? Were any of the questions that scored particularly high or low closed after the August 2012 data was released? What happened to the interesting questions over time?

Unused attributes. There were several available attributes and tables in the data set that we did not use in this study. For example, Stack Overflow allows questions and answers to be tagged with keywords. We did not look at whether or how posts varied in language or source code usage between different tags. We also did not look at user-specific variation, demographic information, or edit counts. User variation would probably be especially fruitful. Some research has already been done in the evolution of expertise on Stack Overflow [12]. Our work could be extended to consider whether high reputation users behave differently in terms of their use of source code and non-source code text. Finally, we did not perform any analysis on *comments* to posts, only on posts themselves (questions and answers).

Size and speed. In the system we designed, querying across so many large tables is fairly slow. Were we to integrate multiple snapshots and currently unused columns and tables into our analyses, the queries would get even slower and more cumbersome. We are open to suggestions, such as using a document-oriented database and search engine technology for some queries.

6.2. Limitations of code-text-ratio algorithms

Accuracy. The analyses shown in sections 5.1-5.3 depend on our accuracy in splitting source code from text. The greatest threat to validity, then, for these sections is that we might have misunderstood text as code, or more likely, we may have missed some code and called it text. The most likely reason this could happen is that the original author may have written source code and not used the Stack Overflow <code> tag within the post editor to demarcate the lines of code from the rest of the post body. The site does have the ability for other users to edit posts to correct things like this (earning reputation for doing so); as such, the system is supposed to be self-correcting. However, there are occasional posts that slip through with code that is not using correct markup.

6.3. Limitations of the FKRE scoring

The main threats to validity in the readability formulae come from mis-identifying sentences, words, and syllables. As mentioned in 6.2, we rely on plaintext only for calculating the FKRE. Any text that is not source code (and marked as such with a <code> tag) is going to be left in the plaintext. This seems appropriate, until we consider the number of “other things” that are in text that are not really code but are not really text either (or at the very least are confusing text that probably should be treated specially). URLs and the content of error messages are the two most common. A user may use a URL as a part of speech, particularly as a noun. An error message may contain extremely long “words” or strange punctuation and very long sentences.

Here is an example of a tricky post to parse for sentence count:

```
I wrote a very short program for an easy
programming competition problem with an online
judger
(http://acm.sgu.ru/problem.php?contest=0&probl
em=184) but for some reason I was getting a
runtime error on the 21st test (it doesn't
specify what exactly the runtime error is.) I
at first tried changing the BufferedReader to
Scanner in the nonworking code and using
in.nextLine() but that didn't work.
```

The sentence counting needs to be sophisticated enough to recognize URLs, un-matching parenthesis, as well as code snippets like in.nextLine() that are not marked up with tags and have internal periods. Also the FKRE formulae do not understand CamelCase such as “BufferedReader” in the example above. This will be considered a single, many-syllable word.

6.4. Limitations of the sentiment scoring

The threats to validity in sentiment analyzing include choosing a bad list of words to start with, misclassifying words as "good" that are really "bad" in a particular domain (or ignoring domain-specific words that should be important). For example, the word list we chose [9] is not domain specific. The word *bug* does appear on the negative list (along with *buggy* and *bugged*) but the synonym *exception,* used in the software development domain, does not appear. (However, *exceptional* and *exceptionally* do appear – on the positive list!) So ideally, general-purpose sentiment word lists will be augmented with important words from the domain.

7. Conclusion

In this paper we performed five simple analyses designed to provide more detail for the general advice given on the Stack Overflow Frequently Asked Questions for authors to include "a bit of source code" when constructing a "good" question.

Although this analysis focuses mainly on the Stack Overflow community of software developers, the results of this analysis may be more broadly applicable to other highly-technical communities, as well as to the production of technical documentation in general.

We found that high scoring questions tend to have a lower code-text ratio than do high scoring answers. We found that among the most favorite questions, lower source code-text ratios were more common. When comparing accepted answers to non-accepted answers, it was more probable that both would have low code-text ratios. But when answers did include source code, accepted answers tend to have more of it. The non-code text portion of the "best" postings tended to be very readable and contain lots of positive words. We recommend that to construct a good posting on Stack Overflow, the "bit of" source code should be higher in answers than in questions, in relation to the non-text portion of the posting. A ratio of about one part of code to every three parts of text (1:3) in answers is ideal, and a ratio of about 1:9 in questions is ideal. We further recommend that the non-code text be geared toward a Flesch reading score of 30-80, with an ideal range of 40-50. To accomplish this, use short sentences and simple words. Finally, the tone of the "best" postings should be positive, and the code and error message portions should be pulled out into <code> blocks to assist with readability (both human and machine).

8. References

[1] Stack Overflow. http://stackoverflow.com

[2] Frequently Asked Questions – Stack Overflow. http:// stackoverflow.com/faq

[3] Non-English Question Policy – Stack Overflow http:// blog.stackoverflow.com/2009/07/non-english-question-policy/

[4] How does reputation work? – Stack Overflow http://meta.stackoverflow.com/questions/7237/how-does-reputation-work

[5] Badges – Stack Overflow. http://stackoverflow.com/ badges

[6] R. Flesch. "A new readability yardstick". *Journal of Applied Psychology*, 32(3):221-233. (1948).

[7] S. Kemper. "Measuring the inference load of a text." *Journal of Educational Psychology*, 75(3):391-401. 1983.

[8] Test your document's readability – Word – Office.com. http://office.microsoft.com/en-us/word-help/test-your-document-s-readability-HP010148506.aspx

[9] http://www.cs.uic.edu/~liub/FBS/opinion-lexicon-English.rar

[10] M. Hu & B. Liu. "Mining and summarizing customer reviews." *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2004),* Seattle, Washington, USA, Aug 22-25, 2004.

[11] Caching - Flash CS4 refuses to let go–Stack Overflow. http:// stackoverflow.com/questions/2193953/flash-cs4-refuses-to-let-go

[12] A. Pal, S. Chang, and J.A. Konstan. "Evolution of Experts in Question Answering Communities". *Proceedings of the Sixth International AAAI Conference on Weblogs and Social Media*. 2012. pp 274-281.