CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Analysing the differences in comment quality between open source development and industrial practices: a case study

Bachelor of Science Thesis in Software Engineering and Management

AHMED NUUR
ALEXANDER GUSTAFSSON
AWELE AZIMOH

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



**Analysing the differences in comment quality between open source development and industrial practices: a case study**

**Many developers view comments as one of the most important artifacts of software development, however few comparisons have been made examining the differences in commenting habits of open source developers and industrial practises. This paper finds differences in the quality of comments between these two categories. The goal of highlighting these differences is to potentially help bridge this gap and improve the internal quality of software products.**



Supervisor: MIROSLAW OCHODEK
Examiner: JAN-PHILIPP STEGHÖFER

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Analysing the differences in comment quality between open source development and industrial practices: a case study


Ahmed Nuur
Department of computer Science and Engineering
Software Engineering & Management Program
Gothenburg Sweden
gusnuuah@student.gu.se

Awele Azimoh
Department of computer Science and Engineering
Software Engineering & Management Program
Gothenburg Sweden
gusaziaw@student.gu.se

Alexander Gustafsson
Department of computer Science and Engineering
Software Engineering & Management Program
Gothenburg Sweden
gusgusalh@student.gu.se



***Abstract*— Many developers view comments as one of the most important artifacts of software development, however few comparisons have been made examining the differences in commenting habits of open source developers and industrial practises. This paper finds differences in the quality of comments between these two categories. The goal of highlighting these differences is to potentially help bridge this gap and improve the internal quality of software products.**



***Keywords—code comments; ; concurrency; Dale-chall ;***


## I. INTRODUCTION

### A. Background

The importance of commenting code cannot be overstated. Many view comments as the second most important artifact used in gaining an understanding of a system, second only to the code they describe. It is also one of the most widely used means of gaining such an understanding.[8]

Few studies have examined the potential differences in quality regarding comments in open source development and industrial practices. One of the potential reasons being difficulties at acquiring source code from the industry.

Commenting code helps improve maintainability and readability[1][2]. Some studies have shown that the commenting habits of open source developers and industrial practices widely differ[3][4], this could mean a difference in comment quality. Finding and remedying such a difference could lead to improvements in the maintainability and readability of future software.

This paper aims to fill a gap in knowledge regarding the differences in the quality of comments. Highlighting these differences could potentially allow for improvements in internal quality of software products. If this paper finds a significant difference in the quality of comments this could allow for future research to investigate potential reasons for

these differences or even a way of eliminating them.

B. Research Questions

**Main Research Question:** What are the differences in quality between code comments in industrial practices and open source development?

**Assumptions:** We assume that there are some differences in the commenting habits of open source developers and current industrial practises. This means that searching for differences in metrics such as comment density are unneeded and focus can instead be placed on comparing specifically the quality of comments.

The remainder of this paper is organised as follows: section II gives an overview of related work, section III describes the research methodology used in the study. In section IV the results of the study are presented and discussed. Section V discusses the threats to validity, section VI summarizes the research findings.

## II. RELATED WORK / BACKGROUND

According to Arafat and Riehle[5] commenting code is a critical part of programming. Also their findings are that both team and project size are unrelated to the comment density of the product whilst the age of the project correlates in a way that the comment density decreases as the project ages. This would help account for certain factors in our study as it would mean that differences in team and project size do not affect the amount of commenting expected. It is worth noting, however that this project only encompassed open source projects and so this may still need accounting for when dealing with the larger software industry. On their approach to arrive to the solution they have used a database from about 10,000 successful open source projects that were active at the time of the study, on the other hand since we are mostly interested in differences in the quality of the comments in open source development when compared to that of industrial practices, this approach will not be appropriate for our study.

Sundbakken[3] finds in his examination of four open source projects a comment density ranging from 0.09% to 1.22%. When examining a closed source project Siy and Votta[4] instead found a comment density of around 50%. This would indicate that there is not only a difference in the commenting habits of open source developers and those working in the industry but also that it is substantial.

Khamis et al. decided on a number of metrics that can be used to determine the quality of comment [6]. One of these being readability heuristics; an example of this being The Fog Index. Then they used these metrics as basis for their analysis of comments using a natural language processing program they developed. Some of these metrics will be good for determining the quality of comments that we aim to examine. This report will be making use of readability as a metric for quality and although Khamis et al. made use of a different metric for readability it could still provide insight into the value of readability. The study also gives information regarding natural language processing and its viability in a study such as this.

Steidl et al.[7] proposed two different metrics to determine coherence attributes, first using words contained in the comment to compare it to the words contained in the method name, second, using the length of inline comments as their indicator of their coherence to the line of the code. This coherence metric will be used in this report to help determine the quality of comments.

## III. RESEARCH METHODOLOGY

To bring us to understand this complex issue we will be using a case study to arrive to a solution. According to Colin Robson (1993) “case study is a strategy for doing research which involves an empirical investigation of a particular contemporary phenomenal within its context using multiple source of evidence”. When dealing with large software companies isolating some issues can be complicated and we deem it infeasible to attempt an experiment in such a short period of time. Quantitative data collection will be used to collect data which are analysed through numerical comparison and statistical analysis. We have developed an automated tool that will help us evaluate the code we are given via several metrics that we have decided upon.

To collect the data, we have approached one software company that was willing to share with us their source code. As for the open source projects data was gathered by

analysing source code from projects that were being actively worked on and that have existed for at least a year. To make sure that projects were actively worked on we looked closely, checking if the amount of the contribution has not fallen dramatically since last year, such as having 30% of last year's contribution. The other major criteria for selecting the open source projects was that it should be written in C++ programming language, this was mainly due to the code we received from the company being written in C++. It is also unclear how selecting projects that were written in a different programming language could have changed the results.

**Evaluation:** Once we have collected the data we measured quality using several metrics. One of these metrics is the coherence coefficient created by Steidl et al [7]. A coefficient like this was deemed useful for determining the usefulness of a comment based on the context it intends to explain. A simple way of describing this metric would be as how different a comment is from the code it describes. If a comment is too similar it can be seen as obsolete and if they are too different it often does not do a good enough job of describing the code[7]. Another metric that was used is the Dale-Chall score in an attempt to measure the readability of the comments as it has been shown to effectively determine the reading difficulty of a given text [12]. We have also considered metrics for both the amount of acronyms used and the length of the comments. Once we have established these values for both the open source projects and the industrial ones we made use of statistical inference testing to make certain any potential differences are scientifically significant. The statistical inference tests used on the study would include the Shapiro-Wilk test to check for normality of data[15] and the Welch Two Sample t-test checking for significant difference between the data sets(concurrency and dale-chall scores for both types of source code)[10][14][16].

The generalizability of the study is difficult to fully estimate but it is worth noting that this report only examines 4 cases. It has been shown that the size of the project and and teams should have no effect on the quality, so they have been eliminated as factors [5]. Using the same programming language and making sure all projects are still being maintained certainly helps, however more studies will most likely have to be conducted. An example of where it is difficult to determine which group to generalize to is in the case of the software companies. It is impossible to determine if the habits we see them exhibit are representative of companies in general or if it just localized to Swedish companies for example.

*Interview Guide*

Upon completion of collecting and analysing the given data, we followed up with an interview session. The interview was conducted in a semi-structured manner. We decided to use this approach because it is highly interactive, researchers can clarify respondents and probe unexpected responses [11]. It is a strategy to avoid one-worded answers. The questions are designed in an open-ended manner which should lead to meaningful and thoughtful responses as opposed to triggering simple yes and no answers. [13]

Interview Questions:

1. A concurrency value greater than 0 but less than 0.5 is considered an acceptable value, one of the comment that we analysed has a concurrency value of 0.75, does the comment add any significance to the understandability of the code?
2. A comment is seen as normal if it has a comment length between 3 to 29 words, one of the code comment that we came a-cross has exactly one word as its comment, does this make any significance to the understandability of the code?
3. A code comment that we have analysed has a concurrency value of 0, does the comment has any importance in regard to understanding the code?

The interview questions were constructed based on our metrics and the results from evaluation of the company's comments. This was done with the intent of getting evaluation for the metrics from the developers' perspective and to identify their reasoning behind why notable comments were done the way they were in some of the cases. For example, a comment is said to be normal if the text is within the range 3 to 29 words; comments that are out of range were noted down for questioning. The interview was conducted after the researchers evaluated the comments from the company's source code. There was not a particular criteria for selecting the subject we interviewed, we simply picked developers of the company based on their availability and the only condition being that they were aware of the code in question. Only one developer was available for the interview,

however he was a senior developer so, his knowledge was well grounded to answer the interview questions regarding the comments in the company's source code.

Before the commencement of the interview, a small preparatory session was conducted for the subject to have an idea on the basis of the interview and an understanding of the kind of interview questions that would be asked. The researchers first introduced the subject to the metrics used for evaluating the comments: Concurrency, Readability (Dale-chall score) and Length of comments, These were thoroughly explained to the subject: making sure he understood the metrics and reasoning behind them. Then the subject was asked of his opinion on the chosen metrics and was made aware that the questions were based on the metrics and the notable findings from the evaluated comments. The subject was rather pleased with the metrics and was comfortable enough to answer the interview questions.

## IV. RESULTS

**Measurement instrument:** The measurement instrument developed is a simple application created using JAVA . The user interface (see Appendix, Figure 3) of the application allows the user to input both a comment and the code it describes and then format the comment. Then it calculates and reports the quality measures, i.e. readability and concurrency (see Appendix, Figure 4). The calculation of the measures is based on natural language processing tools (NLP). We use Stanford Core NLP for that purpose. [9]. Stanford Core NLP is a Java framework for natural language processing that allows us to process the comments and perform the tasks such as sentence and word segmentation. To calculate the readability measure, we firstly segment the comments into words and then calculate its Dale-Chall score. The score is calculated by comparing the words of the comment to a list of 3000 words that at least 80% of the American 4th graders tested knew when read.[10] The percentage of words outside the list is a good indicator of the difficulty of the text. The higher the percentage of words used that are not included in the list the higher the level of reading comprehension needed to understand the text becomes. Concurrency is calculated by dividing the amount of similar words in the comment and code with the total amount of words in the comment. To determine if two words are similar the application calculates the Levenshtein distance (LD) between them and if it is two or lower the application considers the words similar. Levenshtein distance is measured by calculating the cost of changing one word into another. The cost increases by one for each character that needs to be changed, removed or added until both words are the same. The application is expected to analyse only the words in comments, so a filter function was implemented. On execution it loops through the comment and formats it by removing the characters or combinations such as: (/, *, //**) usually used in declaring a block or line of comment in code.

**Comment-quality metrics:** The results collected from the company consists of 42 C++ functions and their comments. These were selected randomly from a set of source codes provided by the company. At least 5 functions and comments were selected from each source-code file of the said set, therefore providing a sample representation of the company's source code. However analysing a larger sample of data would potentially result to a more sufficient data for a baseline and consequently improve the generalizability of the research.

The results show an average concurrency of 0.5. As most comments that have a score higher than 0.5 can be seen as redundant this could be indicative of a high number of comments that add little of value. The average Dale-Chall score is 7.77 which means that on average the comments should be easily understood by an average 9th or 10th-grade student [14]. If we instead of using the average of the scores from each comment use all the comments to calculate the combined score the result gives us a value of 8.64 which is supposed to be easily understood by an average 11th or 12th-grade student [14]. The average length of the comments is 9.78.

The results collected from actively worked open source projects are also 42 C++ function and their comments, they exhibit an average concurrency of 0.32 which is slightly lower than the average concurrency collected from the software company and is thus considered an acceptable value[7]. The average Dale-Chall score is 9.66 which also shows that the average comment should be easily understood by an average 9th or 10th grade student[14]. The average length of the comments collected from the open source projects are 8.2. A comment length is seen as normal if its not less than 3 or more than 29 words, however some of the code comments that we have examined has a length less than

3 which implies that the comment contains less information or explains the obvious.

**Interview:** After finishing the development of the application, the researchers visited the company to collect data. At said company we were given access to several files containing commented C++ source code. We analysed the data on the spot and after this we interviewed one of developers first trying to confirm that our metrics were useful and second of all asking questions about parts of the code that our analysis indicated were poorly commented.

**Statistical inference testing:** To further compare the metrics Welch's T-test[16] is used on both the Dale-Chall score and concurrency between open source and the examined company. This will show if the findings occurred simply by chance. A T-test was chosen primarily due to the small sample size and also the popularity of the test which means that the results are easy to understand if there is to be further research done in the future. The test assumes a confidence interval of 95% which indicates a 5% chance of encountering a false positive. For the t-test to be usable the data must be assumed to be normalized. The way this is achieved is through use of the Shapiro–Wilk test. The Shapiro-Wilk test returns a probability value(p-value) p. The data can be assumed to be normalized if:

$p < \alpha$

In this case α is 0.05 as the chosen confidence interval was 95%. The Shapiro-Wilk test returns a p-value of 0.02853 for the readability data and a p-value of 0.01259 for the concurrency data both of which are below α and so the data can be assumed to be normalized and so a t-test can be performed.

The t-test returns a t-value t, a p-value p and two critical values $t_{c1}$ and $t_{c2}$. The following hypotheses have been devised to examine the difference in concurrency.

$$H_0 : Ccoef_{Closed} = Ccoef_{Open}$$
$$H_1 : Ccoef_{Closed} \neq Ccoef_{Open}$$

$Ccoef_{Closed}$ represents the concurrency coefficient for the software company and $Ccoef_{Open}$ represents the concurrency coefficient for the open source projects. $H_0$ represents the concurrency values of both being equal. When running the t-test we can determine that $H_0$ is false if $t \neq 0$. However to prove that these findings are significant the value of t must also be compares to the critical values. The findings are significantly different if:

$t <= t_{c1}$ or $t_{c2} <= t$

```
> t.test(DataOS$concurrency, DataCompanies$Concurrency)

        Welch Two Sample t-test

data:  DataOS$concurrency and DataCompanies$Concurrency
t = -3.9399, df = 76.914, p-value = 0.0001781
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.28600545 -0.09396245
sample estimates:
mean of x mean of y
0.3205476 0.5105316
```

*Figure 1.*

As can be seen in figure 1 the value of t is -3.9399, the value of $t_{c1}$ is -0.28600545 and the value of $t_{c2}$ is -0.09396245. As $t < t_{c1}$ the samples are significantly different. The findings are significant if:

$p < \alpha$

In this case α is 0.05 as the chosen confidence interval was 95%. As can be seen in figure 1 p = 0.0001781 and so the findings are significant and so $H_0$ can be ruled out.

The same process can then be applied to the readability score of the comments.

$$H_0 : DaleChall_{Closed} = DaleChall_{Open}$$
$$H_1 : DaleChall_{Closed} \neq DaleChall_{Open}$$

$DaleChall_{Closed}$ represents the Dale-Chall scores of the software company's comments and $DaleChall_{Open}$ represents the Dale-Chall scores of the open source project's comments.

```
> t.test(DataOSS$`Dale-Chall score`, DataCompanies$`Dale-Chall score`)

        Welch Two Sample t-test

data:  DataOSS$`Dale-Chall score` and DataCompanies$`Dale-Chall score`
t = 1.9, df = 80.832, p-value = 0.061
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.08931653  3.87142632
sample estimates:
mean of x mean of y
 9.662833  7.771778
```

*Figure 2.*

As seen in figure 2 t = 1.9, $t_{c1}$ = -0.08931653 and $t_{c2}$ = 3.871142632. As $t_{c1}$ < t > $t_{c2}$ these find samples are not significantly different. As the confidence interval of 95% was used the value of α is 0.05 and as can be seen in figure 2 the value of p is 0.061 which means that the findings are not significant.

## V. VALIDITY THREATS

*Internal validity*

The artifact is developed according to metrics the researchers subjectively deemed optimal for evaluating the quality of comments. There are several other methods which could be used for the evaluation asides the selected metrics, however using the use of other metrics could yield a different set of results for the same sample data. Given the very limited time provided for the research, it was conducted using a small sample data, performing the same analysis over a larger sample of data could lead to different set of results. Moreover if more time was given for the research, additional metrics could have been added for the comment evaluation to make a more comprehensive research. As far as software is concerned, the tool is expected to function without faults on every execution. To minimize the risk of errors due to software related issues / bugs, thorough testing of the software will be performed. The researchers evaluated 42 functions and methods from both open source projects and the chosen company. The data's sample size is not significant enough to be generalized on a larger scale.

*External validity*

The interview was conducted with only one subject due to unavailability of the company's developers. Though the available subject is a senior developer and is well grounded on the company's source code, interviewing one person could give room to bias answers as we're limited to one point view rather having multiple non-externally influenced answers of the various subjects.

## VI. DISCUSSION

Our study shows a difference between comment quality in Open source projects and Industrial projects. Based on the research findings, there was a significant difference in the concurrency. Company projects had a score of 0.5 while open source projects had a score of 0.32. Concurrency is said to be acceptable when it is within the range of values greater than 0 and less than or equal 0.5. In both cases we have values within the range, so they are both acceptable. Moreover the difference in score is a result of comments in the open source data having a higher similarity with their corresponding method. To evaluate readability, using the Dale-Chall score, the results show that the company project has an average score of 7.77 whilst the open source projects has an average score of 9.66. According to the scale set by Dale-chall, both scores fall within the difficulty levels of grade 9 to 15 students, which means the words used in text are familiar and overall the comments can be easily understood by grade 9 to 15 students. However, there is a difference in score, with open source projects having a higher value (9.66) with a difficulty level: easily understandable by grade 13-15 (college students) and the company project having a lower value (7.66) with a difficulty level: easily understandable by grade 9-10 students. Though both results have good readability scores, the difference in score simply shows that on average, the experience level needed to easily understand the comments in company projects is less than that of the open source projects. The researchers analysed the comments of 42 methods in both open source and company data. In total, the company data contained 359 words of comment with an average of 8.54 words per method and open source projects contained 307 words of comments with an average of 7.3 words per method. In both cases, the length of the comments is seen as normal as they are within the range of 3-29 words per code comment.

*Interview*

The focus of the interview was to evaluate the metrics from the developers perspective and to identify the reasoning behind why notable comments were done the way they were. The point of analysis in this case was concurrency and length of comment. The developer's comments about concurrency were in agreement with our metric which suggested that any comment unrelated to the code would yield a concurrency value greater than 0.5. The developer stated the that comment was unrelated to code as confirmed by the concurrency value of 0.75. Also the developer's comment about length of comment agree with the metric which suggest that comments contain words less than 3 and greater than 29 are not normal. The developer stated that the comment is unnecessary and does not improve understandability of code as confirmed by the comment with a length of 1. Overall the responses on the interview show that the developers shared the same opinion as researchers and agreed with the metrics. (see Appendix, Table 4).

## VII. CONCLUSION

The aim of this study has been to identify the potential differences in the quality of comments comparing open source developers with industrial practices. This was accomplished using the application developed in the process that would evaluate the quality of comments based on several selected metrics. In the end the study finds that there is a significant difference in the comment quality of open source developers and industrial practices concerning concurrency. A significant difference in readability however, could not be established.

This study manages to further build upon the studies of Sundbakken[3] and Siy and Votta[4] which give data regarding the comment density of open source developers and those working in the industry. It accomplishes this by making use of metrics created by Steidl et al.[7] to measure the quality of comments.

Future research could focus on the reasons behind the differences in quality found in this report. First investigating if these differences serve a purpose such as projects in the industry needing a higher comment concurrency value. If no good reason for these differences is found then future research could focus on finding ways of amending this disparity.

## REFERENCES

[1] J. Börstler, B. Paech "The Role of Method Chains and Comments in Software Readability and Comprehension—An Experiment" IEEE

transactions on software engineering, VOL. 42, NO. 9, September 2016

[2] T. Tenny "Program Readability: Procedures Versus Comments" IEEE transactions on software engineering, VOL. 14. NO. 9, SEPTEMBER 1988

[3] M. Sundbakken, "Accessing the maintainability of C++ source code" December 2001

[4] H. Siy, L. Votta, "Does The Modern Code Inspection Have Value?"

[5] O. Arafat, D. Riehle "The Comment Density of Open Source Software Code" in Proceedings - International Conference on Software Engineering May 2009

[6] N. Khamis, R. Witte, and J. Rilling, "Automatic Quality Assessment of Source Code Comments: the Javadoc Miner," ser. NLDB '10, 2010.

[7] D. Steidl, B. Hummel, E. Juergens" Quality Analysis of Source Code Comments" ICPC May 20-21 2013.

[8] N. Anquetil, S. C. B. de Souza, K. M. de Oliveira, "A Study of the Documentation Essential to Software Maintenance," ser. SIGDOC '05, 2005.

[9] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, S. McClosky "The Stanford CoreNLP Natural Language Processing Toolkit" In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.

[10] E. Dale, J. Chall "Readability revisited: The new Dale-Chall readability formula." May 1st, 1995.

[11] G. Calkli, Lecture 5 "Data collection techniques" slide 8, change of software development process, 2017, University of Gothenburg

[12] J. Begeny, D. Greene "Can readability formulas be used to successfully gauge difficulty of reading materials?"

[13] T. J Rapley "The art(fulness) open-ended interviewing: some considerations on analysing interviews" December 1st, 2001

[14] http://www.readabilityformulas.com/new-dale-chall-readability-formula.php

[15] Shapiro, Samuel Sanford, and Martin B. Wilk. "An analysis of variance test for normality (complete samples)." Biometrika 52.3/4 (1965): 591-611.

[16] Derrick, Ben, Deirdre, Toher & Paul White. "Why Welch's test is Type I error robust." University of the West of England, Bristol, England

**APPENDIX**

Table 1.

| Method | Concurrency | Dale-Chall score | length of comments | Parameters/return types | Comments | Line of code |
|---|---|---|---|---|---|---|
| GrpcLibrary Codegen init | 0.28 | 10.71 | 7 | | // To call grpc_init(). | 73 |
| std::shared_ptr<ChannelCredentials> SslCredentials | 0.43 | 15.19 | 7 | yes | // Builds SSL Credentials given SSL specific options | 78 |
| GrpcLibrary Codegen init | | | | | // To call grpc_init(). | 80 |
| std::shared_ptr<ChannelCredentials> AltsCredentials | 0.14 | 15.19 | 7 | | // Builds ALTS Credentials given ALTS specific options | 93 |
| std::shared_ptr<CallCredentials> GoogleComputeEngineCredentials | 0.37 | 7.95 | 8 | | // Builds credentials for use when running in GCE | 111 |
| std::shared_ptr<CallCredentials> ServiceAccountJWTAccessCredentials | 0.75 | 11.68 | 4 | yes | // Builds JWT credentials. | 118 |
| std::shared_ptr<CallCredentials> GoogleRefreshTokenCre | 0.8 | 10.16 | 5 | yes | // Builds refresh token credentials. | 133 |

| | | | | | | |
|---|---|---|---|---|---|---|
| dentials | | | | | | |
| std::shared_ptr<CallCredentials> AccessTokenCredentials | 0.8 | 7.02 | 5 | | // Builds access token credentials. | 142 |
| std::shared_ptr<CallCredentials> GoogleIAMCredentials | 0.66 | 14.25 | 3 | yes | // Builds IAM credentials | 149 |
| namespace | 1 | 19.38 | 1 | _ | // namespace | 234 |
| SecureAuthContext cpp_channel_auth_context | 0.11 | 9.62 | 27 | | // const_cast is safe since the SecureAuthContext does not take owndership and the object is passed as a const ref to plugin_->GetMetadata | 242 |
| w->thread_pool_->Add | 0.33 | 9.02 | 3 | | // Asynchronous return. | 212 |

| | | | | | | |
|---|---|---|---|---|---|---|
| std::shared_ptr<ChannelCredentials> CompositeChannelCredentials(const std::shared_ptr<ChannelCredentials>& channel_creds, const std::shared_ptr<CallCredentials>& call_creds) | 0.48 | 10.23 | 52 | | // Combines one channel credentials and one call credentials into a channel composite credentials. //Note that we are not saving shared_ptrs to the two credentials passed in here. This is OK because the underlying C objects (i.e., channel_creds and call_creds) into grpc_composite_credentials_create will see their refcounts incremented. | 158 |
| SecureAuthContext cpp_channel_auth_context(const_cast<grpc_auth_context*>(context.channel_auth_context), false); | 0.5 | 8.26 | 30 | | // const_cast is safe since the SecureAuthContext does not take owndership and the object is passed as a const ref to plugin_->GetMetadata. | 242 |
| GRPC_METADATA_CREDENTIALS_PLUGIN_SYNC_MAX | 0 | 11.58 | 2 | | // Synchronous return | 258 |

Table 2.

| Comment | Methods | concurrency | Dale-chall words | length of comments |
|---|---|---|---|---|
| Very unlikely: it requires 2^32 distinct threads to wait simultaneously | bool cond_variable::imp_wait(u32 _old, u64 _timeout) noexcept verify(HERE), _old != -1; | 0.46 | 9.11 | 13 |
| TLS variable for tracking owned mutexes | thread_local std vector<shared_mutex*> g_tls_locks; | 0.2 | 14.4 | 6 |
| Acquire writer lock | imp_wait(m_value.load()); | 0.33 | 9.01 | 3 |
| Convert to reader lock | s64 value = m_value.fetch_add(c_one - c_min); | 0.25 | 7.75 | 4 |
| Wait as a reader if necessary | if value + c_one - c_min < 0<br><br>NtWaitForKeyedEvent nullptr, int*&m_value + 1, false, nullptr; | 0.5 | 6.55 | 6 |
| Acquire writer lock | imp_wait(0); | 0.0 | 9.0 | 3 |
| Convert to reader lock | m_value += c_one - c_min; | 0.25 | 7.75 | 4 |
| Convert to reader lock | s64 value1 = m_value.fetch_add(c_one - c_min); | 0.25 | 7.75 | 4 |
| Wait as a reader if necessary | while futex int* &m_value.raw() + IS_LE_MACHINE, FUTEX_WAIT_BITSET_PRIVATE, int value1 >> 32, nullptr, nullptr, INT_MIN | 0.5 | 6.55 | 6 |
| If blocked by writers, release the reader lock and try again | const s64 value2 = m value.fetch_op[] s64& value | 0.25 | 6.84 | 12 |
| Check reader count, notify the writer if necessary | if _old + c_min % c_one == 0 | 0.2 | 5.8 | 9 |
| Load new value, try to acquire c_sig | const s64 value = m_value.fetch_op([](s64& value) | 0.37 | 7.9 | 8 |

| | | | | |
|---|---|---|---|---|
| Conditional decrement | return m_value.fetch_op([](s64& value) { if (value >= c_min) value -= c_min; }) >= c_min; | 0.0 | 19.4 | 2 |
| Conditional decrement (TODO: obtain c_sig) | return m_value.compare_and_swap_test(c_one, 0); | 0.37 | 17.77 | 8 |
| Try hard way | const s32 value = m_value.op_fetch([](s32& value) | 0.0 | 0.14 | 3 |
| Use sign bit to acknowledge waiter presence | if value && value > INT32_MIN<br><br>value--;<br><br>if value < 0 | 0.28 | 6.2 | 7 |
| Remove sign bit | value -= INT32_MIN; | 0.0 | 0.14 | 3 |
| Signal other waiter to wake up or to restore sign bit | futex & m_value.raw(), FUTEX_WAKE_PRIVATE, 1, null pointer, null pointer, 0; | 0.36 | 7.03 | 11 |
| Conditional decrement | const s32 value = m_value.fetch_op([](s32& value)<br>{<br>if (value > 0)<br>{<br>value -= 1;<br>}<br>});<br><br>return value > 0; | 0.0 | 19.4 | 2 |
| Check RTM and MPX extensions in order to filter out TSX on Haswell CPUs | static const bool g_value = get_cpuid(0, 0)[0] >= 0x7 && (get_cpuid(7, 0)[1] & 0x4800) == 0x4800;<br>return g_value; | 0.21 | 11.0 | 14 |

| | | | | |
|---|---|---|---|---|
| Check AVX512F, AVX512CD, AVX512DQ, AVX512BW, AVX512VL extensions (Skylake-X level support) | static const bool g_value = get_cpuid(0, 0)[0] >= 0x7 && (get_cpuid(7, 0)[1] & 0xd0030000) == 0xd0030000 && (get_cpuid(1,0)[2] & 0x0C000000) == 0x0C000000 && (get_xgetbv(0) & 0xe6) == 0xe6;<br>return g_value; | 0.37 | 13.26 | 16 |
| Magic static | static asmjit::JitRuntime g_rt; | 0.5 | 0.099 | 2 |
| Memory manager mutex | shared_mutex s_mutex; | 0.33 | 14.25 | 3 |
| Size of virtual memory area reserved: 512 MB | static const u64 s_memory_size = 0x20000000; | 0.333 | 11.0 | 9 |
| Try to reserve a portion of virtual memory in the first 2 GB address space beforehand, if possible. | static void* const s_memory = []() -> void* | 0.5 | 9.3 | 20 |
| Reset memory manager | extern void jit_finalize() | 0.0 | 9.0 | 3 |
| Helper class | struct MemoryManager : llvm::RTDyldMemoryManager | 0.0 | 0.09 | 2 |
| Verify address for small code model | if ((u64)s_memory > 0x80000000 - s_memory_size ? (u64)addr - (u64)s_memory >= s_memory_size : addr >= 0x80000000) | 0.0 | 9.11 | 6 |

Table 3.

| Method | Concurrency | Dale-Chall score | length of comments | Parameters/return types | Comments |
|---|---|---|---|---|---|
| [redacted] | 0.3478260867 | 7.619943478 | 23 | yes | [redacted] |

| | | | | | |
|---|---|---|---|---|---|
| [redacted] | 0.375 | 8.33955 | 16 | yes | [redacted] |
| [redacted] | 0.5 | 11.68 | 4 | yes | [redacted] |
| [redacted] | 0.5 | 13.8458 | 8 | yes | [redacted] |
| [redacted] | 0.5 | 9.33 | 16 | yes | [redacted] |
| [redacted] | 0.43 | 7.6685 | 21 | yes | [redacted] |
| [redacted] | 0.714 | 10.712271 | 7 | _ | [redacted] |
| [redacted] | 0.666 | 9.16743 | 6 | yes | [redacted] |
| [redacted] | 0.6 | 0.248 | 5 | yes | [redacted] |
| [redacted] | 0.4 | 0.24 | 5 | yes | [redacted] |
| [redacted] | 0.5 | 5.9957 | 8 | yes | [redacted] |
| [redacted] | 0.8333 | 0.2976 | 6 | yes | [redacted] |
| [redacted] | 0.6666 | 9.167 | 6 | _ | [redacted] |
| [redacted] | 0.3043 | 8.1124 | 23 | _ | [redacted] |

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

| | | | | | |
|---|---|---|---|---|---|
| [redacted] | 0.4444 | 12.805 | 9 | _ | [redacted] |
| [redacted] | 0.6 | 10.1645 | 5 | _ | [redacted] |
| [redacted] | 0.333 | 9.0186 | 3 | _ | [redacted] |
| [redacted] | 0.333 | 9.3162 | 9 | yes | [redacted] |
| [redacted] | 0.4545 | 9.8911 | 11 | yes | [redacted] |
| [redacted] | 0.5 | 13.8458 | 8 | _ | [redacted] |
| [redacted] | 0.4 | 8.8425 | 10 | _ | [redacted] |
| [redacted] | 0.5 | 13.5525 | 10 | No | [redacted] |
| [redacted] | 0.56 | 12.81 | 9 | _ | [redacted] |
| [redacted] | 0.375 | 11.883 | 8 | Yes | [redacted] |
| [redacted] | 0.3684 | 11.1894 | 19 | Yes | [redacted] |

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

| | | | | | |
|---|---|---|---|---|---|
| [redacted] | 0.647 | 5.403 | 17 | yes | [redacted] |
| [redacted] | 0.8 | 9.6138 | 15 | Yes | [redacted] |
| [redacted] | 0.66 | 0.45 | 9 | yes | [redacted] |
| [redacted] | 0.347 | 8.11 | 23 | | [redacted] |
| [redacted] | 0.44 | 12.805 | 9 | | [redacted] |
| [redacted] | 0.5 | 11.68 | 4 | | [redacted] |
| [redacted] | 1 | 9.018 | 3 | | [redacted] |
| [redacted] | 1 | 0.19 | 4 | | [redacted] |
| [redacted] | 0.4 | 8.43 | 21 | Return missing | [redacted] |
| [redacted] | 0.5 | 0.248 | 5 | | [redacted] |
| [redacted] | 0 | 0.1984 | 4 | | [redacted] |
| [redacted] | 0.6 | 7.025 | 5 | | [redacted] |
| [redacted] | 0.66 | 6.5507 | 6 | | [redacted] |

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

| | | | | | |
|---|---|---|---|---|---|
| [redacted] | 0.6 | 0.248 | 5 | | [redacted] |
| [redacted] | 0.5 | 5.54 | 12 | | [redacted] |
| [redacted] | 0.25 | 5.995 | 8 | | [redacted] |
| [redacted] | 0.333 | 9.167 | 6 | | [redacted] |
| | 0.5105315735 | 7.77177844 | 9.785714286 | | |

The method and comment section of the company result table is highlighted in black to conceal the company data. They company requested that we kept the information confidential hence the highlight

**ARTIFACT**

Repository:https://github.com/huphup68/Comment-Quality-Evaluator

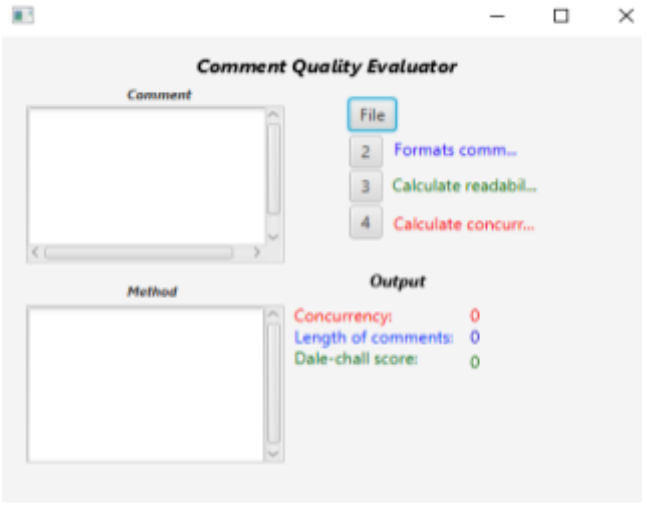


*Figure 3.*

Figure 3 is an image of the Comment quality Evaluator tool that has been developed. After execution, the data: comments and method are inputted for the application to analyse and return results.

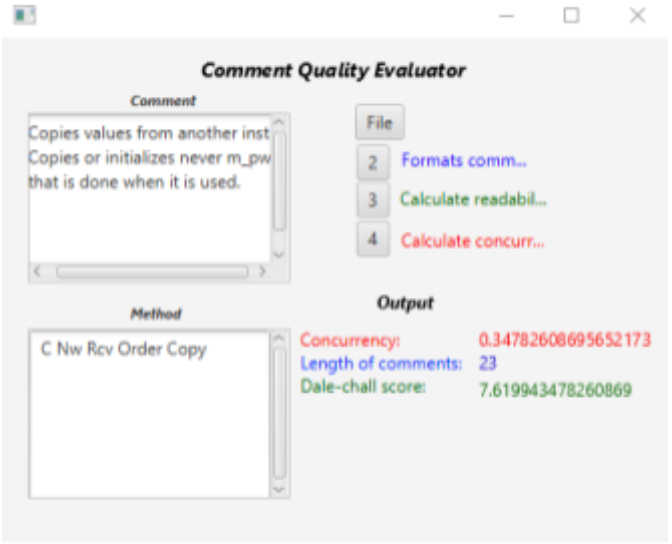


*Figure 4.*

Figure 4 is an image of the tool after analysing the given data. The concurrency value is higher than 0 and lower than 0.5 and is therefore considered an acceptable value [7]. The length of the comment is not less than 3 nor more than 29 which means the comment is seen as normal. The Dale-Chall score of the comment indicates that it should be easily understood by an average 9th or 10th-grade student [14].

Table 4.

| | Question | Answer |
|---|---|---|
| Q1 | A concurrency value greater than 0 but less than 0.5 is considered an acceptable value, one of the comment that we analysed has a concurrency value of 0.75, this means the code comment is almost similar to the function name, does the comment add any significance to the understandability of the code? | I would have prefered to have a comment that says something about the method function, however that's not the case here & its difficult to see exactly what one would have written. |
| Q2 | A comment is seen as normal if it has a comment length between 3 to 29 words, one of the code comment that we came a-cross has exactly one word as its comment, does this make any significance to the understandability of the code? | It's a keyword that describes one of the variables in the code, I can see what it's referring to, however its unnecessary and does not improve the understandability of the code. |
| Q3 | A code comment that we have analysed has a concurrency of 0 value, does the comment have | The comment does not add anything to the understandability of the code, in the comment the author |

| | | |
|---|---|---|
| | any importance in regard to understanding the code? | simply credits the original contributor of the code and has re-used the code. |

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018