Objective Correlation Metrics for Quality of Code Estimation

[*]Note: Please remove notices before uploading paper. [4]

Eleftheria Chatziargyriou
Higher Institution of very nice code
second floor, apartment 3

Konstantinos Kanavouras
Lower Institution of very nice code
Python Script Execution Engineer

***Abstract*—Code quality is really important for writing code which is of great quality. In this paper we are trying to formalize a way in which code quality is easily assessed so bosses from all around the world can easily distinguish the 100000000×ers from the 0.0000000333333Xers.**

I. Introduction

Why do we write code? Of course, to implement the features. The more code we write, the more features we implement.

However, instead of writing just the code, we must explain what the code does, in so-called "commit messages". Some commit messages are good at explaining a lot of code, while some are bad at explaining less code.

In this research, we identify:

- What is a good commit message?

We also ask:

- Is there a link between good commits and more features?

And finally:

- Which programming language is the best?

II. Quality Assurance of Commit Messages

A. Commit Message

When Linux Torvalds famously said "I hope you all die" [6], he was referring to the quality of bad commit messages he was forced to read back when he created the famous operating system Linux named after himself (Linus).

Some developers write simplistic comments such as *"Fix compiler warning"*. This offers no real information to the user as questions may arise such as "what compiler warning did you fix?" and "how did you fix said compiler warning?". Users who aren't lazy could instead write *"Fix compiler warning C4842"*.

It is evident that the above approach leaves a lot to be desired. For one, it doesn't indicate how said warning is fixed which is really what a good engineer should be focusing on. The authors would recommend a more detailed commit message such as: *"Fix compiler warning C4842 by not using offsetof and simply counting the bytes at a glance"*.

This is undoubtedly an improvement over our previous, lousy commit message. However, a watchful reader will point out that we still rely on outside information for parsing a commit. Beginner developers who have yet to memorize all compiler warnings by heart may not know what compiler warning `C4842` refers to. Of course, this is a subset of real developers, but for the sake of being welcoming to newcomers, we could transform our commit message into *"Fix compiler warning C4842 (the result of 'offsetof' applied to a type using multiple inheritance is not guaranteed to be consistent between compiler releases) by not using offsetof and simply counting the bytes at a glance. This warning was introduced in Visual Studio 2017 version 15.3 (compiler version 19.11.25506.0). For more information refer to https://docs.microsoft.com/en-us/cpp/error-messages/compiler-warnings/compiler-warnings-c4400-through-c4599?view=msvc-170."*. We assert that the more words a commit message has, the more quality the message has. This is summed up in the following theorem:

Theorem 1: If a commit message (in English) cm_1 has many distinct words and another commit message (also in English) cm_2 has a less or equal amount of distinct words and it's also true that $cm_2 \subseteq cm_1$, then $q(cm_1) \geq q(cm_2)$, where q is the quality-counting function.

To prove the above theorem, take the set of distinct words that make up the commit message cm_2, U_2. Now, take the set of distinct words that make up the commit message cm_1, U_1. We assume that $U_1 \subseteq U_2$. Let QA be a relation of a set of words in W (all the sets of all the words ever but only in English) to its quality in Q (the quality of all words).

Then the image of U_1 by QA is a subset of the image of U_2 by QA [9]. In plain English $QA[U_1] \subseteq QA[U_2]$ which means that the quality of the message will always increase or at the very least remain the same the more words you add.

This theorem has many practical applications to the real world. Namely, this indicates that a developer can add as many words as they can until they reach the utmost quality which has a theoretical maximum of plus infinity and a theoretical minimum of zero.

A reader may wonder why the theoretical minimum is set at zero and not at some other cooler number like $-\infty$. This happens because the quality of something can be zero (no quality) but it can't be negative. To prove this statement consider a negative quality. But then the quality is so low that it wraps around and it begins to become high again

and thus we're back at positive quality [11] which leads to a contradiction. Thus no quality can be less than zero. ■

An observant reader may be quick to point out that the above theorem is only true for English. Extending the theorem to more languages is not within the scope of this work. However, interested readers are invited to contact the authors in case they desire to delve deeper into this exciting field of never-ending possibilities such as finding the objectively best language for commit-message-writing.

We would like to bring to the attention of the readers, the fact that some more authoritarian version control systems restrict the number of words one may use, forcing most messages to be of lower quality.

B. Commit Message Word Quality

An ultra observant reader may also note that this only proves that the more words we add the better the quality becomes. However, switching words means that the precondition $cm_2 \subseteq cm_1$ does not hold, and word count alone cannot obviously determine the quality of the commit.

To make our point even more clear think of the word "Disrupt" and think of the word "Bread". It should be evident that the world "Disrupt" has more quality than "Bread", but according to *Theorem 1*, "Disrupt Bread Production" is at least of equal quality to "Disrupt". The question we aim to answer is whether "Whole wheat bread" is of more, less or equal quality to "Disrupt".

It should be obvious that the motivation for something to be of quality should be to find what is *useful* as opposed to what is *not useful*. A word is useful if it gives you an important lesson for life in general [1]. Therefore, we can give the following very utilitarian definition:

Definition 2.1: The quality of a word w, $q_{S_W}(w)$ is the maximum number of points a player can acquire if they play it on a Scrabble board S_W. S_W is the maximum-scoring board containing the words $W = (w_1, w_2, ...w_n)$ played consecutively.

Scrabble is a very important game that has preoccupied great minds for years [2] so it makes sense to hold words that can score better at a higher standard.

TABLE I: Scoring of Letters in Scrabble [10]

Points	Letters
1	A, E, I, O, U, L, N, S, T, R
2	D, G
3	B, C, M, P
4	F, H, V, W, Y
5	K
8	J, X
10	Q, Z

Note that by definition, words longer than 7 letters are excluded since you can't play them on the first round unless you are cheating and your opponents are painfully oblivious to the rules. If a word has more than 7 letters, then we just discard the other letters. We also arbitrarily decided to not use score modifiers because they are lame and the authors can't be really bothered at this point. Our last utterly random decision, is that invalid words are completely ignored - if a word can't be played on a given board, it's also ignored.

We can finally define the following definition:

Definition 2.2: The commit message quality *cmq* of a message *msg* of number of words N, is defined as $cmq(msg) = q_{S_{(w_1,...w_{N-1})}}(w_N)$.

Fig. 1: Disruptive Bread Production

III. Determining Quantity of Code

As mentioned in Page, the functionality of code is only affected by the amount of code written. Since ancient years, the Source lines of code metric (SLOC) has been used to meter the code, including developer salaries, benefits and watercooler discussion time allocation.

However, the lines of code metric has been heavily contested by various people [3]. That is why we propose a new, objective, language-agnostic metric for code quantity, called "characters of code" or COCO for short.

It is obvious that the intertwining of whitespace, comments and other so-called "human-readable" glyphs disturb the accuracy of the COCO metric. Therefore, we will define a further number of metrics to assess the core soul of our program:

- *CCOCO*: Comment characters of code
- *CCOCO*: Clear characters of code (space, tabs, weird UTF-8 invisible characters)
- *SCOCO*: Symbolic (symbols) characters of code
- *COCOCO*: Concrete (heavy) characters of code, defined as:

$$\text{COCOCO} = \text{COCO} - \text{CCOCO} - \text{SCOCO} - \text{CCOCO} \quad (1)$$

Of course, in the interest of *TODO: think of something clever to add here*, we can define multiples of the code quantity metrics to aid conversation for "engineers" who can't be bothered with exact numbers. This paper uses the word-renowned IEC 80000-13 [5] standard for representation of units, for example:

- 1 *kiCOCO* = 1024 characters of code
- 1 *miCOCO* = 0.0009765625 characters of code

For example, let's look of the code quantity of the following Br**nfuck snippet:

```
>++++++++[<+++++++++>-]<.>++++[<+++++++>-]<
+.+++++++..+++.>>++++++[<+++++++>-]<++.----
--------.>++++++[<+++++++++>-]<+.<.+++.----
--.--------.>>>++++[<++++++++>-]<+.
```

The COCOCO metric is:

$$\text{COCOCO} = 164 - 0 - 0 - 164 = 0 \tag{2}$$

Obviously the result is 0 miCOCO, since Br**nfuck is not really code now is it.

IV. Actual Data (yes)

We all know that no one reads the theory, we just included some pages of nonsense to give the impression we know what we're talking about and to distract from the fact that our reasoning is ultimately flawed. However, none of this should matter if we provide some fancy .

We wrote some *code* to calculate the word quantity and quality as we defined in Sections II and III. For this, we scrapped the most starred projects on GitHub (which stores at least half the code in the world [8] so it's representative according to the Law of Much Code). We analysed 52 122 commits from the top 50 repositories of every language for some statistics persuasion confidence.

In Figure 7 we can look at pretty plots that honestly don't say much about our data but they are still nice to look at.

We see that those two parameters can't be correlated linearly, it is most likely due to the data points that don't fit our assumption and are thus deemed invalid. However, it also becomes immediately obvious that most people don't bother to write neither good commit messages nor many COCOCO. The authors urge Chief Human Efficiency Engineers all around the world to incorporate those two revolutionary metrics as a way to weed out the leech-developers that suck the productivity out of any team.

52122

Fig. 2: Sample size

Code quantity and commit quality were also analysed individually producing some results that are at the very least marginally visually appealing. We used an advanced tool [7] to visualise the commit quality as it is demonstrated in Figures 3 and 4.

While in Figure 7, there isn't any significant visible difference between different programming languages, in Tables II and III, some trends can already be seen in the languages and ideas that make programmers write good or bad commit messages. AI (Artifical Intelligence) is obviously doomed to bring more hatred towards your keyboard and computer, while high level languages and irrelevant refactoring fixes that don't really matter towards the grand scheme of things given the small size of our planet Earth relatively to the rest of the observable universe make developers spend more effort documenting their commits.

TABLE II: Some of the *best* commits we encountered

Commit message	Language	Scrabble Score
Rename 'proto_package_to_prefix-_mappings_path' to 'package_to_prefix-_mappings_path'.	C++	137
Make it more clear in package.json which keys will be included in published packages (#1846)	TypeScript	128
Instantiate models with new Model instead of Object.create(model) for performance.	JavaScript	124
Merge remote-tracking branch 'origin/master' into crawler-process-reactor-later	Go	111
fixed a stupid compiler error, because MutableList and ArrayList have different constructors (yeah Interfaces can have constructors – they can't)	Cobol	91
fix(athena): Fixes export bucket location. Fixes column order. (#4183) * ok * ok * ok * ok * ok	Rust	88

TABLE III: Some of the worst commits we encountered

Commit message	Language	Score
YOLOv3 ON CPU!!!!	C	2
It's actually pronounced YOLOYOLOYOLO	C	2
[xiami] xiami is dead	Python	2
no opencv bad opencv	C	2
functions in cobol	Cobol	2
Numeronym in cobol!	Cobol	2
Impl 'std::Error' for 'serde::json::Error'.	Rust	4
infoschema: stabalize TestSelectClusterTable test (#33295)	Go	4

Undoubtedly, this wouldn't be a real paper if we didn't objectively assess our own code as well. From Table IV, it is obvious that our commit quality far outweighs the average of every language. The same goes for our code quantity, which is good. This is definite proof that the authors of this paper are qualified to speak about what happens in the code. In fact, we were so proud of the software written for this paper, that we decided to share it with you, at https://github.com/kongr45gpen/objective-commit-parser. It includes an automatic Scrabble mangler, a typical armanda of dataviz packages and many unidentified and identified bugs that cause serious flaws in the output.

TABLE IV: Average commit statistics for our code

Commit quality	46.83
Code quantity	2469.83

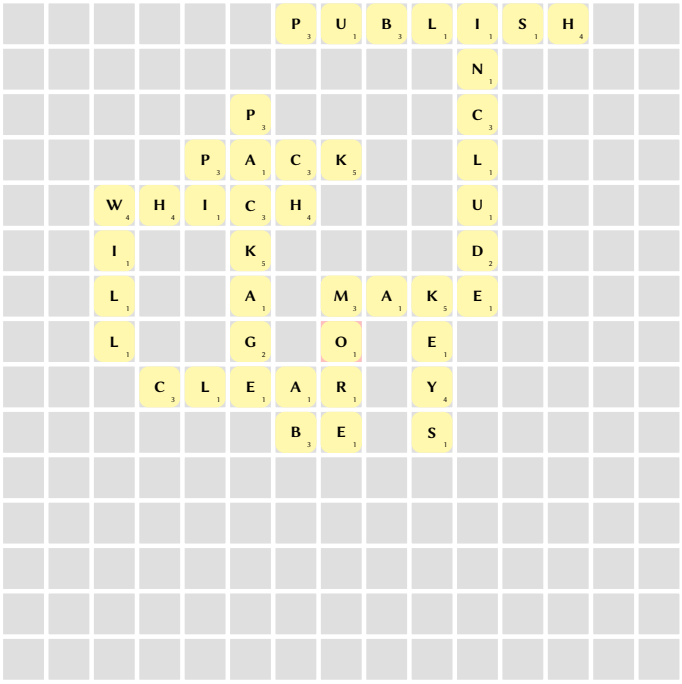

Fig. 3: Auto-generated best Scrabble board for commit *Make it more clear in package.json which keys will be included in published packages (#1846)* with score 128

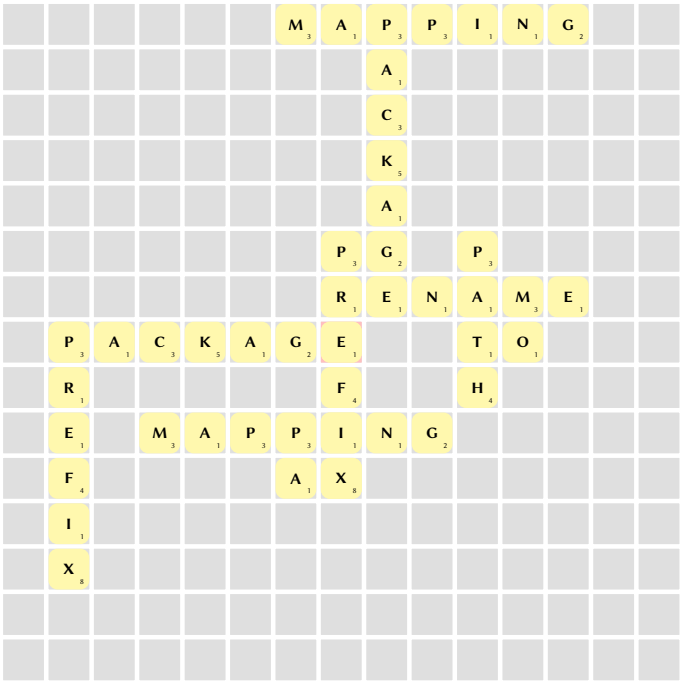

Fig. 4: Auto-generated best Scrabble board for commit *Rename 'proto_package_to_prefix_mappings_path' to 'package_to-_prefix_mappings_path'.* with score 137

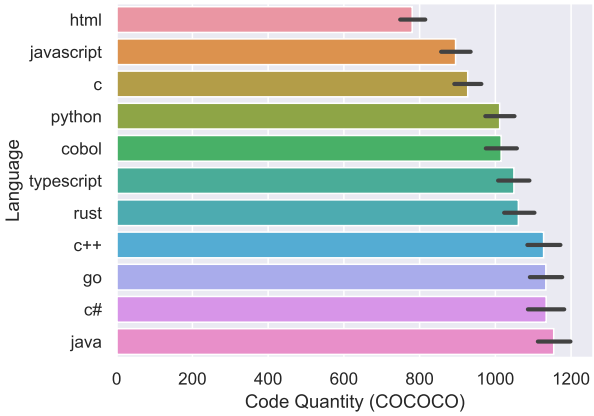

Fig. 5: Comparison of COCOCO code quantity metrics. It is obvious that large Java class names prevail over anything related to web development.

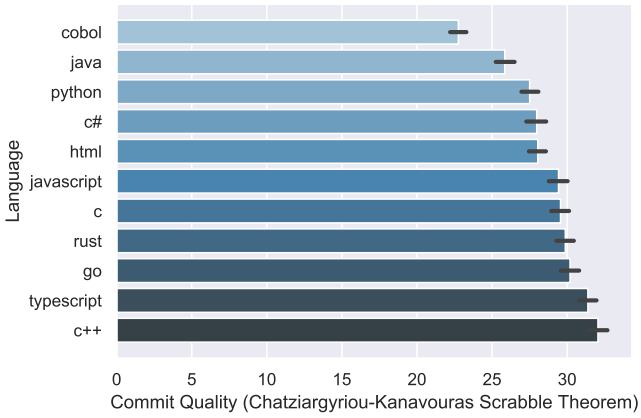

Fig. 6: Comparison of commit message quality. Compiled languages and TypeScript are so hard that commits have to explain what's going on.

V. Conclusions

In this paper we've derived some very objective metrics for estimating the quality of code through the commit messages and the actual code output. In all honestly, neither of us thinks that any of this makes much sense. We kinda made things up as we went along really. After all, we leave out the most important of all the productivity-assessing parameters which is time spent on stand-up meetings vs time wasted on the toilet.

Regarding languages, from Figures 5 and 6 it is obvious that C++ has both the best commit messages, and a very high standing in COCOCO metrics. On the other hand, HTML developers don't really write a lot, and Cobol developers don't document their changes adequately. We therefore unanimously declare C++ as the best language. ■

Nevertheless, we honestly believe that our work makes significant strides towards a more fair and egalitarian developer community.

References

[1] Wikepedia contributors (all of them). *Microsoft Word*. URL: https://en.wikipedia.org/wiki/Microsoft%5C_Word.

[2] Dr. Tom Murphy VII Ph.D. "What Words Ought to Exist?" In: Special Interest Group on Harry Q. Bovik 2011. Apr. 1, 2011. URL: http://tom7.org/papers/sigbovik2011tom7whatwords.pdf.

[3] *How Bad Is SLOC (Source Lines of Code) as a Metric?* Stack Overflow. URL: https://stackoverflow.com/questions/3769716/how-bad-is-sloc-source-lines-of-code-as-a-metric.

[4] Nicolas Hurtubise et al. "Refutation of the "Failure to Remove the Template Text from Your Paper May Result in Your Paper Not Being Published" Conjecture". In: Special Interest Group on Harry Q. Bovik 2021. Apr. 1, 2021, pp. 297–299. URL: http://sigbovik.org/2021/proceedings.pdf.

[5] *Iec 80000-13 - Google Search*. URL: https://www.google.com/search?q=iec+80000-13.

[6] *Linus-Eff-You-640x363.Png (PNG Image, 640 × 359 Pixels)*. URL: https://cdn.arstechnica.net/wp-content/uploads/2013/02/linus-eff-you-640x363.png.

[7] Mark. *Scrabble Visualizing Tool*. URL: https://tex.stackexchange.com/a/194797.

[8] Linux mostly and some others as well. *Linux*. URL: https://github.com/torvalds/linux.

[9] James Munkres. *Topology*. 2nd edition. Upper Saddle River, NJ: Pearson College Div, Jan. 7, 2000. 537 pp. ISBN: 978-0-13-181629-9. URL: https://www.amazon.com/Topology-2nd-James-Munkres/dp/0131816292.

[10] Robby Findler. *Scrabble Rules*. 2008. URL: https://users.cs.northwestern.edu/~robby/uc-courses/22001-2008-winter/scrabble.html.

[11] YourMovieSucksDOTorg, director. *YMS: Neil Breen*. Dec. 13, 2017. URL: https://www.youtube.com/watch?v=6L4g3H_TM28.

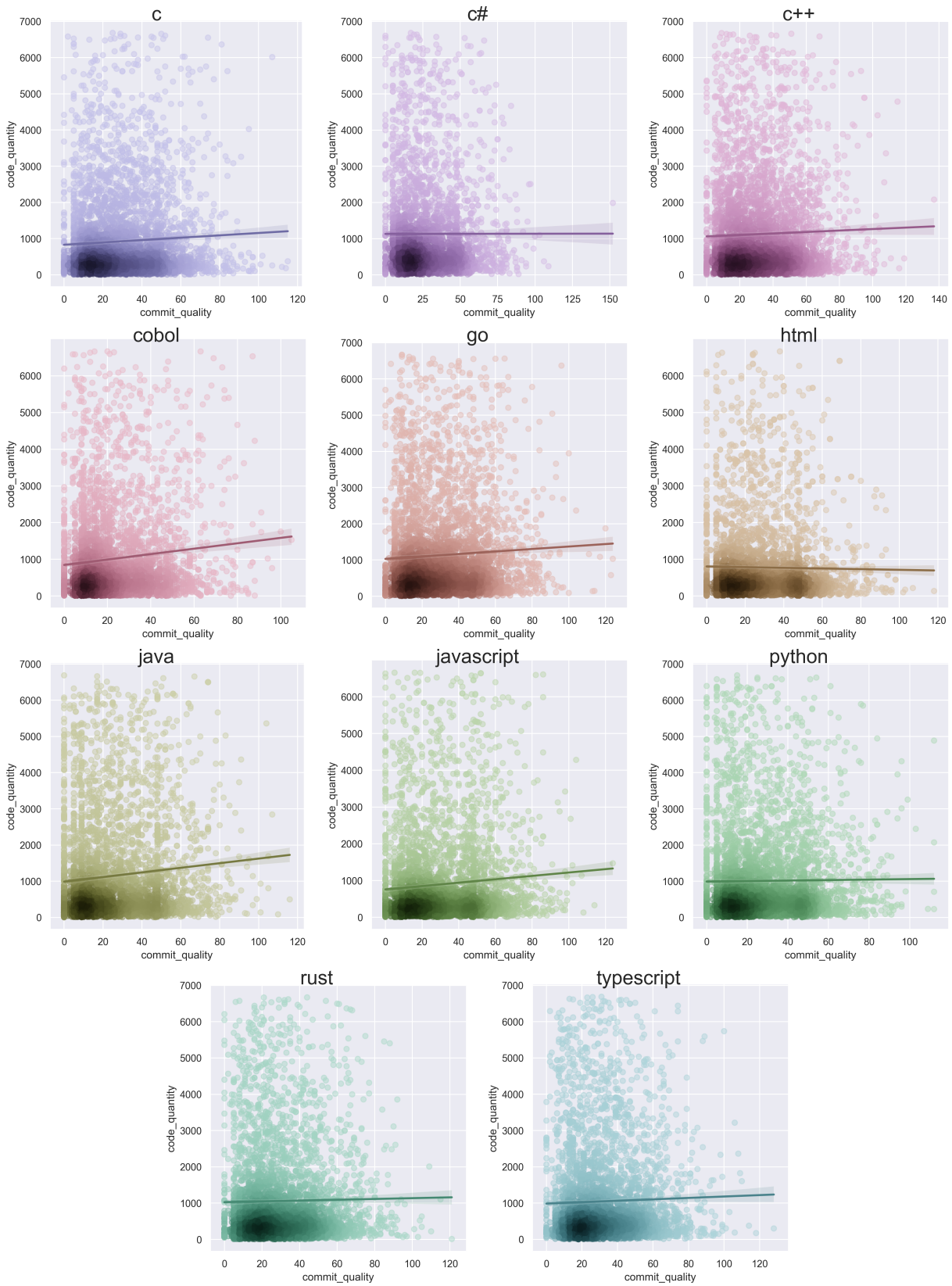

Fig. 7: Correlation code quality/quantity