Code Review Comments: Language Matters

Vasiliki Efstathiou and Diomidis Spinellis
{vefstathiou,dds}@aueb.gr
Department of Management Science and Technology
Athens University of Economics and Business
Athens, Greece

ABSTRACT

Recent research provides evidence that effective communication in collaborative software development has significant impact on the software development lifecycle. Although related qualitative and quantitative studies point out textual characteristics of well-formed messages, the underlying semantics of the intertwined linguistic structures still remain largely misinterpreted or ignored. Especially, regarding quality of code reviews the importance of thorough feedback, and explicit rationale is often mentioned but rarely linked with related linguistic features. As a first step towards addressing this shortcoming, we propose grounding these studies on theories of linguistics. We particularly focus on linguistic structures of coherent speech and explain how they can be exploited in practice. We reflect on related approaches and examine through a preliminary study on four open source projects, possible links between existing findings and the directions we suggest for detecting textual features of useful code reviews.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; *Information extraction*; • **Software and its engineering**; • **Social and professional topics** → *Software maintenance*;

KEYWORDS

Collaborative Software Development, Code Review, Natural Language Processing, Lexical Semantics

ACM Reference Format:
Vasiliki Efstathiou and Diomidis Spinellis. 2018. Code Review Comments: Language Matters . In *ICSE-NIER'18: 40th International Conference on Software Engineering: New Ideas and Emerging Results Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183399.3183411

1 INTRODUCTION

Archived communications from developer interactions provide a rich source of information which, under appropriate analysis and interpretation, may reveal interesting facts related to practices, decisions and team dynamics in a collaborative development environment. Software peer review is a QA practice widely adopted by open source and commercial projects [24] where, besides finding defects, it offers additional benefits such as knowledge transfer across members of a development team [3]. To this end, a number of qualitative studies have been conducted towards identifying factors that influence code review usefulness including, among others, the textual characteristics of the review comments [3], [24], [14]. Other studies have combined empirical and text mining methods, to identify linguistic aspects of constructive code review comments [6], [23]. However, the existing studies are limited to superficial textual characteristics such as stop word ratio and sentence length, with no semantic interpretation. Current research for deriving semantics from natural language exchanged by developers is largely directed towards the extraction of behavioural aspects which may only indirectly affect the progress of a project [10], [21]. Related studies applying sentiment analysis research with software engineering data are heavily based on off-the-shelf tools, which are trained in different contexts, such as product or movie reviews and may therefore not apply to other domains. In addition, although in sentiment analysis research the objective is to mine opinions related to specific artefacts and the valence of emotions triggered by these artefacts, adaptations in software engineering arbitrarily assume a *unidirectional* causal relationship from negative team emotions to negative effects on the performance of a collaborative project. That is to say, the effect that a toxic project may have on the dynamics of the development team is not taken into consideration. Consequently, negative results regarding the suitability of applying state of the art sentiment analysis methods on software engineering research have started to emerge [12]. Ongoing research works towards filling this gap by collecting domain-specific software engineering data [22], [19], or performing more fine grained classification of emotions in collaborative software development [9].

In this work we investigate aspects of linguistic semantics that go beyond emotion and opinion valence detection. Working on the hypothesis that communication in a collaborative environment primarily concerns conveying factual objective information rather than expressing subjective opinions, we investigate facets in language which may indicate cues of effective collaborative messages. We focus our study on code reviews due to their value in software development and the increasing interest of the community for improved practices [3]. We believe however that the outcomes from the directions we propose are transferable to other types of communications related to software development and maintenance, such as bug reports, issue tracking comments or informal chats.

2 RELATED WORK

The study by Rahman et al. [23] is the one that shares objectives closest to ours. The authors acknowledge the importance of effective code review comments, point out the lack of related automated text mining techniques and attempt to interpret findings from empirical studies into textual properties of comments [6], [14]. Taking into account the change-triggering effect of a review as a heuristic for usefulness, they extracted a number of features from the text of the

ICSE-NIER'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5662-6/18/05…$15.00
https://doi.org/10.1145/3183399.3183411

review comments. Their study however is limited to structural characteristics of the text with no consideration of interpreting the underlying semantics. Based, for instance, on the finding from Kononenko et al. [14] that *'Clear and thorough feedback is the key attribute of a well-done code review'* they study clarity on the basis of reading ease, defined in terms of word and sentence length. Reading ease and question ratio in comments did not provide significant results in their study. However other dimensions did, with most prominent being the presence of actual source code elements quoted in the review text and the '*conceptual similarity*', a metric that captures the proportion of vocabulary shared between the review comment and the corresponding modified source code. Another factor taken into account was the stop word ratio, which proved to be higher in non-useful comments, albeit with a small effect size. Based on the derived textual and other features related to developer expertise, the authors built a model for predicting review comment usefulness.

Similarly, Bosu et al. [6], based on the findings of an interview-based empirical study, built a decision tree classifier for distinguishing between useful and not useful code review comments. They used attributes related to reviewer expertise, change set and revision thread metrics and textual features of the review. The textual features they used comprised a vocabulary of 349 keywords that were frequently found in comments that had been classified as useful through the empirical study.

In this study we investigate exclusively linguistic information contained in the text of the comment. The benefit of assessing a review by its comment is that it does not require information that can only be available upon completion of the review, such as thread and change set metrics mentioned in the previous paragraph. Looking at the comment text alone, offers opportunities for tools classifying a comment at real-time and recommending improvements before its submission. From the features mentioned in this section the ones that are solely language-based and gave promising results are the existence of source code elements in the comment [23] and the set of keywords mentioned by Bosu et al. [6].

3 FUNCTIONAL SEMANTICS OF LANGUAGE

Peer code review is by definition a judgmental process; it involves reviewing and assessing whether a change suggested by a peer is appropriate for merging into the codebase. As such, the related communications have to reflect the reasons why a change needs to be made and consequently, reasons for accepting or rejecting this change. The need for appropriately justified code review comments has been identified in the literature under the notions of rationale articulation [3], [25] or thoroughness of the review feedback [14]. In this section we propose directions for grounding these findings in longstanding linguistic theories of rhetorics and discourse coherence [18]. The idea is that there exist implicit relations between the sentences of a text so that the content of one sentence might provide, for example, elaboration or explanation for the content of another. These relations bind a text together, thus contributing to its overall coherence, and they are often made explicit by the speaker through the use of particular cue phrases. In Table 1 we give some examples of such phrases along with their functional semantics as described by Knott and Dale [13]. The cue phrases in Table 1 are examples extracted from a comprehensive vocabulary compiled in by Knott and Dale [13], which has served as the basis for deriving sets of more complex

Functionality	Cue phrases
Causality	Because, since, as, thus, as a result
Contrast	But, however, although, on the other hand
Exemplification	For example, for instance
Clarification	In fact, actually, in other words, in short
Similarity	Also, as well, similarly, too
Hypothesis	If, in this case, assuming that, provided that

Table 1: Examples of coherence relations

lexico-syntactic patterns for mining instances of justifications, analogies and conditional statements in text [17], [20], [26]. These types of structures are context-independent and have been studied across diverse domains such as legal reasoning [2], medical decision support [11], policy making [4], product reviews [27] and lately for discourse analysis in social media [8]. We argue that a similar approach is a promising direction for assessing the linguistic characteristics of useful code reviews. We motivate our approach by revisiting examples from existing work [23] and provide a preliminary analysis on a set of code reviews from four open source projects [28].

4 USEFUL COMMENTS REVISITED

Based on the assumption that stop words carry *'very little or no semantics'*, Rahman et al. worked on the hypothesis that a high ratio of stop words in text is a characteristic of non-useful code reviews. Stop words are words frequently occurring in natural language and although there is no universal list, state of the art vocabularies of stop words include: *a, also, as, but, by, so, the, while* among others. In settings such as document retrieval or topic modelling, they are often removed because their presence in a text does not contribute in distinguishing one concrete topic in text from another. However, even though they are topic-agnostic, in the role of coherence cue words they are necessary to construct higher level semantical meaning, which may be important for articulating accurate comments. This is demonstrated in the single concrete example of useful code review comment presented by Bosu et al. [6]: *'I don't think we need 2 ways to call* `get_partner_whitelabel_config` *as* `market_id` *is* `None` *by default.* Even though the authors acknowledge that this comment may be useful due to the warranted action it expresses, the textual feature they locate as indicative of usefulness is the presence of source code artefacts. The source code itself though would possibly be of little use if it had not been an object of a justification phrase expressed by the term '*as*'. However, it is common practice in text mining studies [6] to remove such words during preprocessing.

Motivated by this example, we investigated the assumption that *source code elements appear often in useful comments because they act as references in explanatory phrases.* To this end, we studied the inclusion of coherence relations in the sets of words that frequently appear in close proximity with source code elements in comments. More specifically, we examined the *collocation* of source code and linguistic coherence elements in the comment messages of code reviews. The objective was to study whether references to source code are often coupled with instances of coherence relations.

For our study we performed a preliminary analysis on publicly available code review data from four open source projects.[1] We selected this data set because it contains a substantial volume of data (~ 250,000 patches) from the well-known open source projects Eclipse, LibreOffice, AOSP and OpenStack, mined from Gerrit code reviews

[1] http://kin-y.github.io/miningReviewRepo/

and organised in a portable database dump [28]. In this analysis we used the textual information of review messages provided in the data set. During preprocessing we did not perform stop word removal or stemming. The only elements we removed were signature strings that were not part of the comment message itself (e.g. 'Author-Id', 'Signed-off-by'). In addition, we replaced non-natural language elements, such as references to URIs, path directories and source code with a dummy string specific to each type of replacement and identical across all comments. We studied the comments of each project separately. For the occurrences of the source code replacement string we generated all the bigram collocations within a window of three words, i.e. all the pairs that contain this string and a word located in the text at a distance of one or two words away, either before or after the string. We counted the occurrences of all distinct words that appear in bigram collocations with the source code string and ranked them in order of descending frequency. In the ranking we excluded the trivial collocations with article 'a(n)' and with instances of other source code elements. The set of coherence relations that we used for this analysis was composed of 100 single-word expressions compiled from the vocabulary given by Knott and Dale [13].

In order to quantify the existence of coherence relations in close proximity with source code in the comments, we computed the inclusion rate of the coherence keyword vocabulary in the top 50, 100, 150 and 200 most frequent source code bigrams in each project. The analysis showed that (a) the top 50 most frequent bigram collocations contain words that denote some coherence relation at a rate of 22%—30% across the four projects, and (b) for bigram collocations of decreasing frequency the inclusion rate of coherence relations tends to decrease. These observations, illustrated in Figure 1, support the intuition that references to source code elements in review comments are often coupled with instances of coherence relations. As indicative examples of coherence relations in this study, we report the coherence keywords most frequently collocated with source code in the four projects: *because, after, or, but, so, instead, since, when, only, if, as, and, for, before, now, not, then, also.* The coherence keywords that were most frequently found in collocation with source code in *all* four projects (i.e. their intersection) are: *as, if, not, for, so, and, also, instead, when.* Note that the keyword *'as'* mentioned in the motivating example appears in the 50 most frequent bigrams of all four projects. Specifically, it appears in the following ranks across all source code bigram collocations of each project: 23^{rd} of 1000 in AOSP, 17^{th} of 400 in Eclipse, 12^{th} of 1000 in LibreOffice and 16^{th} of 3000 OpenStack.[2]

The results of this study motivate further, fine-grained analyses that will (i) include complex lexico-syntactic patterns of linguistic coherence beyond single-word expressions (ii) investigate the involvement of references other than source code in coherence relations (iii) characterise the presence of coherence relations of diverse functionalities (e.g. causality, exemplification) in review comments. We envision methods for automatic evaluation of useful comments that will incorporate features of linguistic coherence.

5 DISCUSSION

The software engineering community has successfully adapted natural language processing methods for identifying textual features of useful code reviews. However, there is a need for establishing synergies with experts who will provide directions for appropriately interpreting these findings and will identify requirements for further studies.

Results from different studies may motivate diverse directions of investigation. In the previous section we presented a possible interpretation of the reasons why a specific code review comment may have been annotated as useful, by examining it from the perspective of linguistic coherence. Bosu et al. [6], mention among other findings that *'verbs of request (e.g., please, should, may be) are more likely to be useful'*. It is worth noting that *'should'* and *'may'* belong to the special grammatical category of modal verbs, which are verbs used to express necessity or likelihood. When interpreted appropriately and examined along with the rest of modals as a category they may reveal aspects of the messages in which they are being used. Moreover, in examples of keywords that appear in useful comments, the authors list among others the verbs *expand, match, remove, move, fix*, whereas with keywords frequent in non-useful comments they list verbs *store, leave, let, keep, work, fail.* Apparently, the useful comments contain mostly verbs that denote some kind of transformation whereas the non-useful comments contain mostly verbs that denote no change in state. Linguists have determined that syntactic behaviour is a reflection of the underlying semantics and have established results in semantic categorisation of verbs [15].

Motivated by this discussion, here are our recommendations for the effective processing of natural language text in software engineering artefacts:

Semantics. Interpret the meaning of text mining results at the *concept* level. Treat words as concepts rather than as plain lexical units; instead of looking at flat lists of results take into account whole conceptual categories in which they may belong.

Syntax and grammar. Do not ignore the lexical semantics hidden in grammatical and syntactical structures. Text preprocessing often removes valuable linguistic information. During empirical studies this information is displayed to annotators who assess the usefulness of comments presented to them in full text. We recommend that models also be developed using the original full text.

Synergies. In order to satisfy the preceding two recommendations we need to consult research outcomes from expert linguists. To avoid re-inventing the wheel we should establish synergies with communities which have already put theory into practice and can contribute with methods [16], [17], [20] and annotated data [5], [7], [1].

6 OPPORTUNITIES

Specifying concrete linguistic features for useful code reviews offers opportunities for advancing the study and practice of code review in diverse dimensions.

Recommendations for good practices. Guidelines for efficient communication coupled with explicit linguistic instructions will not only increase comprehension for the receivers of the messages, but also for the transmitters. Prompting developers to explicitly argue over a suggested change forces them to expose the reasoning to themselves first before they post a comment. This practice could lead to the reduction of instances of non-useful messages such as *'when a reviewer incorrectly indicates a problem in the code, perhaps due to lack of expertise'* [6].

Tools. Good practices can be reinforced through the use of tools which, based exclusively on linguistic features, can provide automated immediate feedback for improvements and prevent non-useful comments from being fed into the review pipeline.

[2] We applied a filter of 10 in the reported totals; words that appear less than 10 times in bigrams with source code elements are not considered.

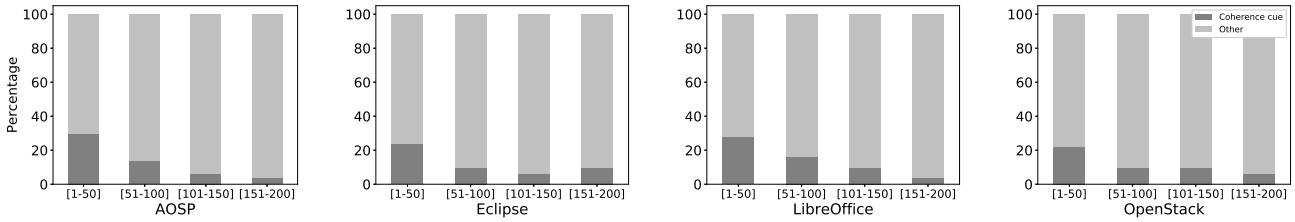

Figure 1: Coherence keyword inclusion in top 200 bigram collocations with source code elements in four different projects.

Archived Data. Code reviews that capture rationale are valuable not only at the project level; they also comprise a rich resource of archived information for later needs, such as retrieval of design rationale [3], [25]. The research ideas we propose contribute towards this direction.

Transferable results. In this study we focused on code reviews. However, the linguistic structures we examined are context-independent. We believe that the proposed approach has potential on a variety of software engineering processes that involve comments not only as part of direct communications but also as source code artefacts. As cited by Kononenko et al. [14], for some developers code quality is about the presence of meaningful comments; '*comments should tell why not what*'.

7 CONCLUSIONS

Code reviews offer rich linguistic information with unexplored directions for research. Text mining studies have provided hints for useful review comments, however it is only with the guidance of expert linguists that these hints can turn into generalised results. Likewise, results from empirical studies on useful comments will be best exploited if interpreted by experts into appropriate linguistic cues. As a first step towards reconciling theory and practice, we proposed adapting linguistic theories of coherence along with text mining for assessing useful code review comments. Our motivation emerged from results in empirical studies that call for thorough well-justified comments as well as indicative examples of useful comments from the literature. Our preliminary case study gave encouraging results for following this direction.

ACKNOWLEDGMENTS

The project associated with this work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732223.

REFERENCES

[1] AIFdb 2017. AIFdb a corpus of analysed argumentation. http://corpora.aifdb.org/. (2017). [last accessed 23-Oct-2017].
[2] Kevin D Ashley and Vern R Walker. 2013. Toward constructing evidence-based legal arguments using legal decision documents and machine learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law*. ACM, 176–180.
[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 international conference on software engineering*. IEEE Press, 712–721.
[4] Trevor Bench-Capon, Katie Atkinson, and Adam Wyner. 2015. Using argumentation to structure e-participation in policy making. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XVIII*. Springer, 1–29.
[5] Luisa Bentivogli, Ido Dagan, and Bernardo Magnini. 2017. The Recognizing Textual Entailment Challenges: Datasets and Methodologies. In *Handbook of Linguistic Annotation*. Springer, 1119–1147.
[6] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 146–156.
[7] Aljoscha Burchardt and Marco Pennacchiotti. 2017. FATE: Annotating a Textual Entailment Corpus with FrameNet. In *Handbook of Linguistic Annotation*. Springer, 1101–1118.
[8] Mihai Dusmanu, Elena Cabrio, and Serena Villata. 2017. Argument Mining on Twitter: Arguments, Facts and Sources. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2307–2312.
[9] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and its direction in collaborative software development. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*. IEEE Press, 11–14.
[10] Emitza Guzman and Bernd Bruegge. 2013. Towards emotional awareness in software development teams. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 671–674.
[11] Anthony Hunter and Matthew Williams. 2012. Aggregating evidence about the positive and negative effects of treatments. *Artificial Intelligence in Medicine* 56, 3 (2012), 173–190.
[12] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering* (2017), 1–42.
[13] Alistair Knott and Robert Dale. 1994. Using linguistic phenomena to motivate a set of coherence relations. *Discourse processes* 18, 1 (1994), 35–62.
[14] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: how developers see it. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 1028–1038.
[15] Beth Levin. 1993. *English verb classes and alternations: A preliminary investigation*. University of Chicago press.
[16] Jiwei Li and Dan Jurafsky. 2016. Neural net models for open-domain discourse coherence. *arXiv preprint arXiv:1606.01545* (2016).
[17] Marco Lippi and Paolo Torroni. 2015. Context-Independent Claim Detection for Argument Mining. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
[18] William C Mann and Sandra A Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse* 8, 3 (1988), 243–281.
[19] Mika V Mäntylä, Nicole Novielli, Filippo Lanubile, Maëlick Claes, and Miikka Kuutila. 2017. Bootstrapping a lexicon for emotional arousal in software engineering. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 198–202.
[20] Raquel Mochales and Marie-Francine Moens. 2011. Argumentation mining. *Artificial Intelligence and Law* 19, 1 (2011), 1–22.
[21] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 303–313.
[22] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele Marchesi, and Bram Adams. 2016. The emotional side of software developers in JIRA. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 480–483.
[23] Mohammad Masudur Rahman, Chanchal K Roy, and Raula G Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 215–226.
[24] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 202–212.
[25] Andrew Sutherland and Gina Venolia. 2009. Can peer code reviews be exploited for later information needs?. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE, 259–262.
[26] Douglas Walton, Christopher Reed, and Fabrizio Macagno. 2008. *Argumentation schemes*. Cambridge University Press.
[27] Adam Z Wyner, Jodi Schneider, Katie Atkinson, and Trevor JM Bench-Capon. 2012. Semi-Automated Argumentative Analysis of Online Product Reviews. *COMMA* 245 (2012), 43–50.
[28] Xin Yang, Raula Gaikovina Kula, Norihiro Yoshida, and Hajimu Iida. 2016. Mining the Modern Code Review Repositories: A Dataset of People, Process and Product. In *Proceedings of the 13th International Conference on Mining Software Repositories*. 460–463.