Analyzing the Relationship Between Commit Messages and Maintainable Software

Kevin Bruhwiler, Jason Stock

1. Introduction

Collaborative development is popular in many applications with notable growth in open source software (OSS). GitHub, a cloud based version control system, is a major supporter with over 260 million repositories and 44 million users, including contributions from community projects, research groups, and commercial companies. GitHub enables developers to seamlessly make code changes and run automated test suites to identify known defects. However, with consistent modifications it is possible for code smells to be introduced unintentionally. Code smells are intended to indicate structural issues within the code that do not impede functionality and go undetected by functional and performance tests. Consequently, they increase the likelihood of introducing bugs and incurring additional maintenance overhead in the future.

Previous works have shown that the emotion of developers' commit messages vary with respect to the time of day, day of the week and team distributions [1, 2]. We extend these concepts in this study by mining popular GitHub repositories to assess how commit sentiment correlates to developer behavior and the design of maintainable code. Having a better understanding of developer trends could bring useful insights to OSS systems and improve the detection of problematic code.

We also measure the information content of the commit messages to determine whether longer and more informative messages are correlated with code quality. Some similar investigations have been performed, such as attempts to measure the amount of "architectural" information in a commit message [3] or determine the "unusualness" of a message in an attempt to predict future anomalies in code [4]. A recommendation system that could advise users on more valuable commit messages or identify potential problems in commits would be very useful for OSS projects with many loosely connected developers.

2. Methodology

We perform a number of sentiment and information content analysis techniques on evolving projects and their respective code changes, with the goal of identifying how code smells are introduced and their relation to the commit messages and specific authors. We aim to gain a greater understanding of how code evolves and is maintained in health and unhealthy ways. All of our source code can be found publicly in GitHub[1].

2.1. Data Collection

There are many commit message data sources for popular Java projects, but few of those contain fine-grained code smell information. For this reason, we identify a variety of different GitHub repositories and carry out a multi-step preprocessing phase to extract commit messages with the specific code smells introduced at every addition.

[1] https://github.com/stockeh/repo-mining-maintainable-code

Repository Identification Large-scale Java projects with an english domain are selected from GitHub's trending webpage. We narrow our search to the repositories containing over 6,000 commits and over 160 contributors. This is to ensure we have enough data from active contributions and obtain a greater breadth of developer interactions. Thereafter, we identify 10 projects from which five are classified as hobbyist and the others as professional to further contrast development practices with an established corporation. As a control for professional projects we explore those from the Apache Software Foundation, including: apache/skywalking, apache/druid, apache/flink, apache/cassandra and apache/beam. Anyone can contribute to an Apache project by means of discussion and proposing features. However, the Apache Way specifies that an individual must first become a valuable contributor before being invited to be a committer with write access to a project. We defined hobbyist projects as OSS with less strict committer requirements with more lenient contributions rules. The following projects that fall in this category are randomly chosen and manually inspected: runeline/runeline, kiegroup/drools, teamnewpipe/newpipe, quarkusio/quarkus and openliberty/open-liberty.

Commit Extraction Commits for each of the 10 projects explored herein are extracted for all added, copied, and modified files. Ignoring all renamed, unmerged, and unknown file changes. A given commit will have a single message and author associated, but may contain multiple changed files. To extract this information we use the command: `git log -p --diff-filter=ACM --full-diff`, with the addition of flags for pretty-printing the output. We generate a .csv file for each project containing the following fields by parsing the formatted log:

- Commit: 20 byte commit hash
- File: name of the file changed with the given commit
- Message: message added with the given commit
- Username: author's GitHub username
- Email: address of the author for the given commit
- Additions: numbers of the lines of code added in the given file
- Deletions: number of the lines of code deleted in the given file

Code Smell Aggregation With knowledge of which files change within a given commit, we are able to extract programming flaws and code smells for specific file versions. This is done by first showing the entirety of the file with `git show` with the specific file and commit, and then running the PMD source code analyzer over the content. Results from PMD identify the line number, description and priority for each of the existing rule violations. This information is then joined to the .csv to correspond violations introduced with the additions made in a file at a given commit. We exclude all non-Java and test classes to only account for source code.

Data Preprocessing Even with GitHub's filtering of spoken language, we found there to be inconsistencies in the commit message. Therefore, all non-english commits are filtered out (to simplify sentiment and information content analysis) and the code smell priority is extracted from PMD's output.

2.2. Analytical Approach

To understand the behavior of developers, we perform sentiment and information analysis on minimally processed commit messages. The techniques described below quantify otherwise qualitative data when code changes are made to a project, enabling the exploration of corresponding commit messages to maintainable code and identifying patterns amongst the authors.

We use three different algorithms for text processing. The first performs sentiment analysis using the Valence Aware Dictionary for sEntiment Reasoning (VADER) proposed by Hutto and Gilbert [5]. VADER is a lexicon and rule-based sentiment classification tool trained on social-media. It interprets the emotion of a short piece of text to be positive, negative or neutral. We use a compound score, a metric between [-1, 1] that represents the normalized sum of the lexicon ratings, as an overall score for a message. We also assess the information content of commit messages using Term-Frequency Inverse-Document-Frequency (TF IDF)[6] and the Lempel-Ziv Complexity [7]. For each message we assign a TF IDF score as the average of individual scores from each word. Less common words will be given a higher score and are assumed to be more informative. Thus, a higher average will represent more unique content of commit messages. Lastly, we measure the Lempel-Ziv Complexity, a Kolmogorov-complexity based method, that compresses the sentence structure under the assumption that text which can easily be compressed contains less information.

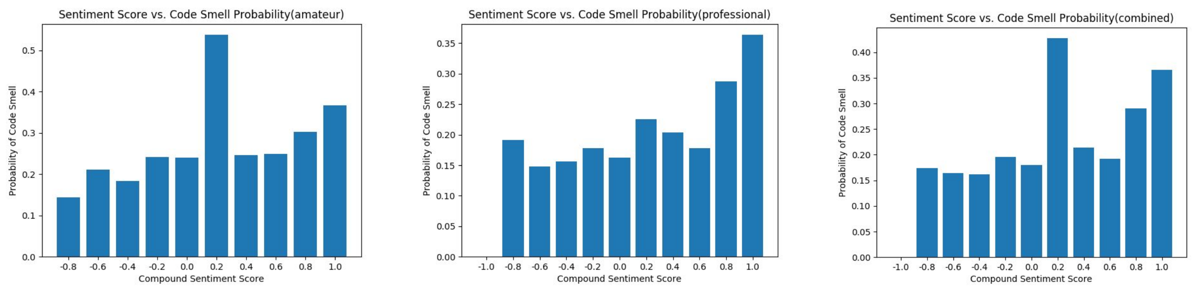
3. Experimental Evaluation

To evaluate our methodology, we compute the sentiment and information content metrics for both the hobbyist and professional projects to contrast the difference in behavior of contributors. The general trends for developers in OSS are evaluated using a combined score from the two categories. We use these metrics and correlate them with the probability of introducing a code smell and its severity into the projects. This removes any bias of an aggregated count as the total for hobbyist projects contain 17267 commits with 1085 unique contributors while the total for professional projects contain 35458 commits with 2033 contributors. Lastly, we investigate how these metrics can identify authors that continuously have a high probability of introducing problematic code.

3.1. Sentiment Analysis

In general, compound scores of projects contain primarily neutral ratings. That is, most commit messages have a sentence structure with values close to zero, which correspond to more informative words such as “fixed” or “hadoop”. We evaluate the probability of having a code smell in each commit to find that smelly code is often introduced alongside messages with more positive meaning (Figure 1).

In order to better understand the type of projects that this observation applies to, we separate the contributions of hobbyist and professional projects and analyze them individually. To this end, we notice the same upward growth of introducing smells with the positive increase of sentiment values in both classes. However, there exists numerical differences between the two that suggest hobbyist projects constitute a greater likelihood of developers adding smells (Figure 1.a).



a) Hobbyist Projects b) Professional Projects c) Combined

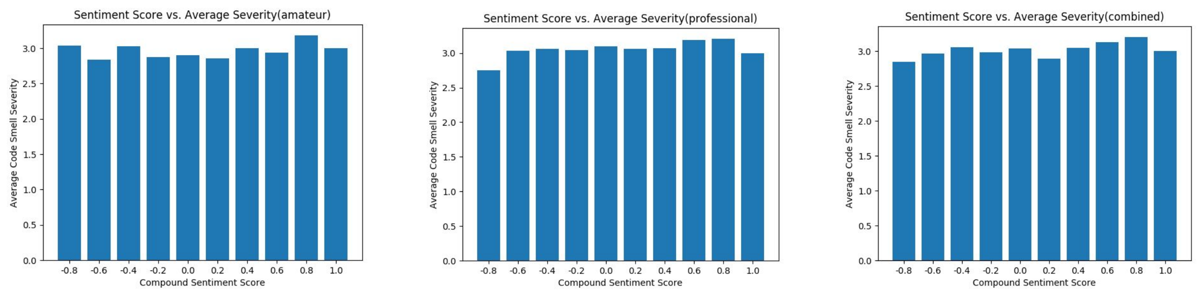
Figure 1: Probability of a code smell existing for observed compound sentiment scores for all projects.

This trend is most likely because words such as “added” or “progressed” carry a positive connotation. While examining the data, we observed that code most likely to introduce a code smell is introduced with commit messages specifying many different feature additions. This makes sense, as additions to the code, especially by developers who are less familiar with the underlying architecture of the project, are more likely to introduce structural issues. Commits where many such additions are included at once are likely to both introduce code smells and be evaluated as emotionally positive. An example of commit messages with different sentiment scores are as follows:

Negative Sentiment *Kill all running tasks when the supervisor task is killed (#7041)*

Positive Sentiment *Set source subscriptions to 'empty' when switching configurations to BASIC. Otherwisethis could cause non-message source events to be processed for output if these sources were enabled before in JSON format configuration. ... - Modify the HPEL Logging Unit Test to use a traceSpecification oof com.* and org.* over the default *=all. Using x.com.*=off did not work. This is to avoid the presence of BufferManagerImpl add() Trace which causes the test to fail. - Keep ThreadLocalHandler's in use in TraceSource. This avoids the triplicate of traces that occur through the collector framework.*

In our preprocessing stage, we use PMD to capture the code smells with their assigned severity ratings. The severity measure, with one as the highest and five being the lowest, brings insight to how problematic the code smell is. For example, a code snippet which contains an unnecessary fully qualified name has low severity as it can clutter the code, but it does not significantly impact maintainability. Whereas violating formal parameter naming conventions can make it difficult to do impact analysis and refactor code. For this reason, we consider how the change of severity is impacted by the sentiment of each message. Figure 2 illustrates our observations for hobbyist, professional and combined results. The average severity fluctuates around 2.8, showing a mix of high and low severity items with no clear trends.



a) Hobbyist Projects b) Professional Projects c) Combined

Figure 2: Average severity of code smells for observed compound sentiment scores across all projects.

3.2. Information Content Analysis

In Figure 3 the information content for every commit message is plotted. It can be seen that lempel-ziv complexity and TF-IDF score have a strong correlation. Both have peaks and troughs in the same locations. However, lempel-ziv complexity appears to be more consistent and is not so strongly influenced by outliers. Analysis of both metrics shows roughly identical trends (discussed below) however TF-IDF introduces more noise. Consequently, we use lempel-ziv complexity in the following analysis.

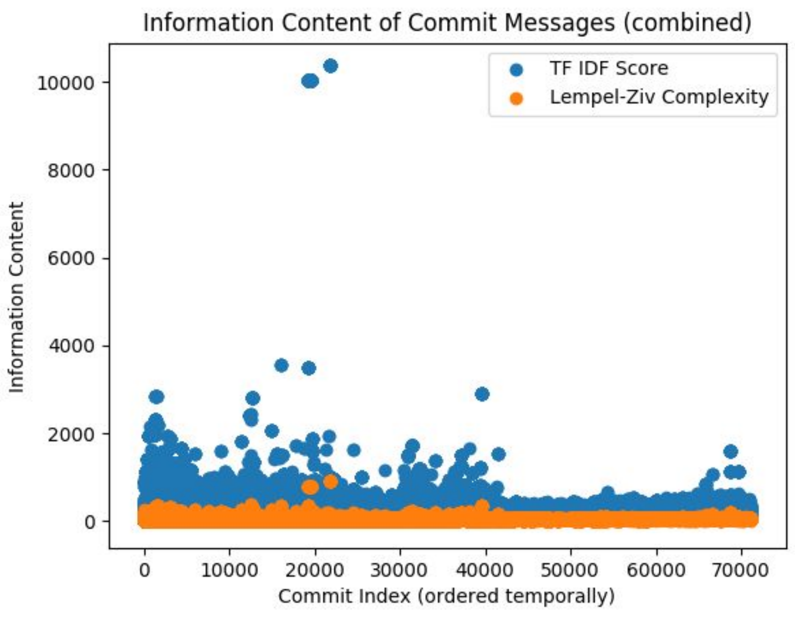
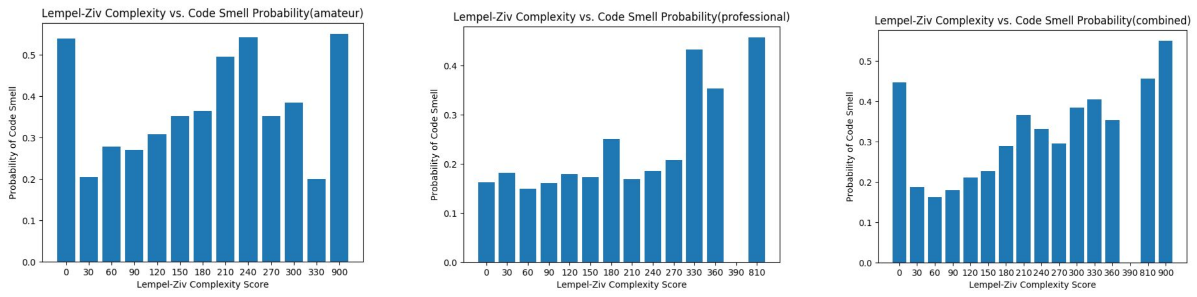


Figure 3: Information content metrics for all the commit messages

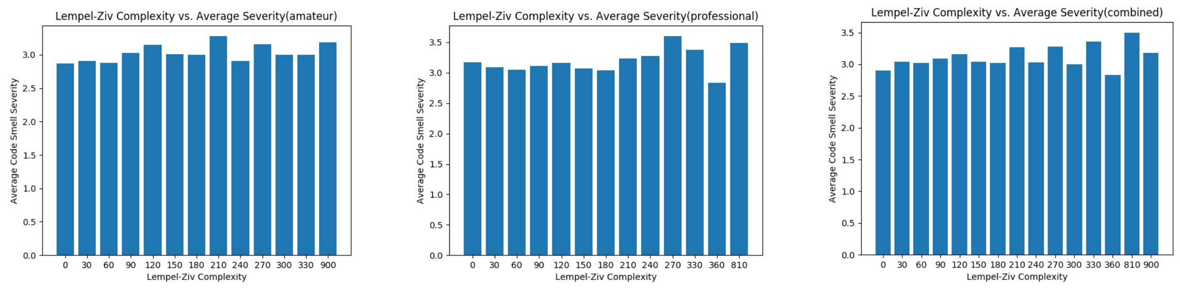


a) Hobbyist Projects b) Professional Projects c) Combined

Figure 4: Probability of a code smell existing for observed Lempel-Ziv Complexity across all projects.

In Figure 4 we see a similar trend to the sentiment analysis, in that more complicated commit messages are more likely to introduce code smells. This pattern holds better for professional projects than for hobbyist projects, however it is visible for both and is especially noticeable when the two datasets are combined. Again, similar to the trends in Figure 1, this is mostly likely due to the fact that complicated commit messages are correlated with a large number of changes. The more changes that are made, the more likely they are to contain structural or stylistic issues.

The very high probability for introducing a code smell for low content commit messages for hobbyist projects is also notable, especially because no such relationship exists for professional projects. In this case, the most likely explanation is that there is a group of novice developers who both do not understand or appreciate the value of commit messages and are inexperienced at writing clean code. Due to the higher standard of contribution for Apache projects, this group of developers are unable to contribute and are consequently not shown in the professional project analysis.



a) Hobbyist Projects b) Professional Projects c) Combined

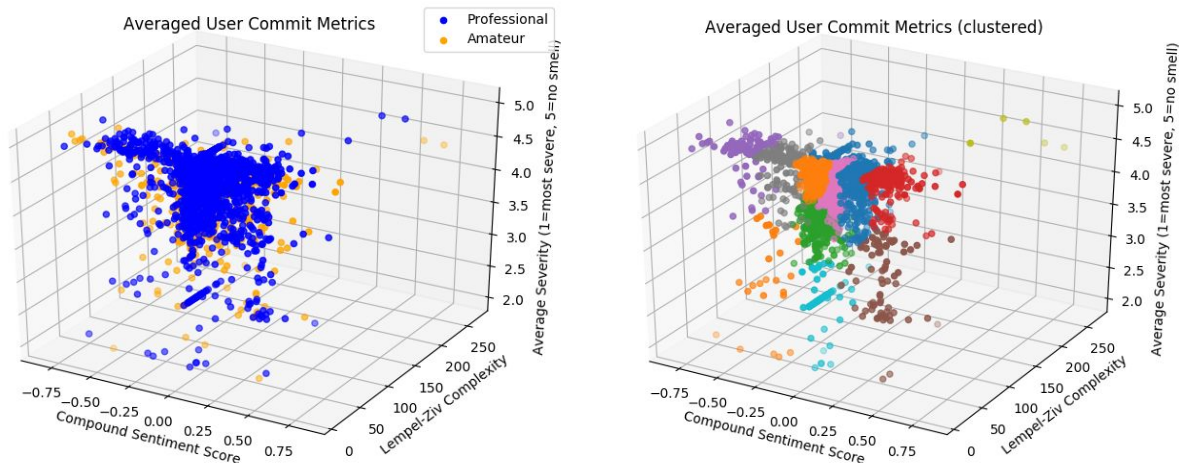
Figure 5: Average severity of code smells for observed Lempel-Ziv Complexity across all projects.

Again, similar to the corresponding sentiment analysis, Figure 5 shows little correlation between information content and code smell severity. The one noticeable feature is that, as messages contain more information, the variability of the code smell severity increases. More complex commit messages may have more severe or less severe code smells, while simpler messages have a much more predictable code smell severity.

It's not entirely clear why this is. Without prior knowledge, one could reasonably predict that more complicated messages are associated with more complicated commits, which are more likely to introduce serious structural flaws. It's possible that these commits are being offset by another group of careful developers who write more detailed commit messages. This hypothesis is partially confirmed in

Section 3.3, as a group of developers who write informative commit messages and are unlikely to introduce code smells can be observed.

3.3. Author Analysis



a) Raw Author Metrics b) Clustered Author Metrics

Figure 6: Averaged user commit metrics for professional and hobbyist projects clustered with k-means clustering.

Figure 6 examines the average values for various metrics for each author, separated by project type and clustered using k-means clustering. Using the within-set sum of squared errors as a function of k, we determine 12 clusters to be optimal for the data via the elbow of the function. It can be observed that most developers write low complexity commit messages and are, on average, unlikely to introduce code smells, regardless of whether they are working on a professional or hobbyist project. In fact, the similarity between developers of the two project types shows little difference in skill or behavior.

We use k-means clustering in a crude attempt to define GitHub developer “personality types”. Partly this is done to examine the hypothesis made in Section 3.2 that there is a group of developers who write complex commit messages and do not introduce code smells (shown here as light yellow in the upper right of Figure 6.b). We also see the hypothesized group of developers who write simple commits and poor code (light orange in the bottom left). There are also some other interesting developer types, such as those who write poor code with very formal commit messages (teal), those who write positive commit messages with poor quality code (brown, likely developers adding features when they don’t understand the underlying architecture), and those who write very negative commit messages with high quality code (purple and grey, these developers are mostly likely troubleshooters, fixing issues in existing code).

4. Conclusions

Open source software has seen immense growth with popular projects having hundreds of developers and thousands of commits. Changes are constantly being introduced to implement new features, fix defects, and improve code. In this study, we assess the impact of these changes on software maintenance and evolution by evaluating the behavior of developers. The most clear conclusion we can make is that complex commits with detailed messages are likely to introduce code smells. It is also interesting to note that these commits do not generally have more severe issues than other commits. We also observe that there is relatively little difference in the quality of code written by developers for Apache projects when compared to other OSS projects on GitHub, despite their quality control measures.

The categorization of different developer “types” based on commit messages may not be especially practical for the development of software, as it is likely easy to identify experienced and novice developers without complex numerical analysis, however it is a useful tool for examining the hypothesis made based on the relationships between commit message sentiment, content, and the associated code quality.

Bibliography

[1] Guzman, Emitza & Azócar, David & Li, Yang. (2014). Sentiment analysis of commit comments in GitHub: An empirical study. 11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings. 10.1145/2597073.2597118.

[2] Sinha, Vinayak & Lazar, Alina & Sharif, Bonita. (2016). Analyzing developer sentiment in commit logs. 520-523. 10.1145/2901739.2903501

[3] Tiago Oliveira Motta, Rodrigo Rocha Gomes e Souza, and Claudio Sant’Anna. 2018. Characterizing architectural information in commit messages: an exploratory study. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering* (*SBES ’18*). Association for Computing Machinery, New York, NY, USA, 12–21. DOI:https://doi.org/10.1145/3266237.3266260

[4] Santos, Eddie A., and Abram Hindle. "Judging a commit by its cover; or can a commit message predict build failure?." *PeerJ PrePrints* 4 (2016): e1771v1.

[5] Hutto, C.J. & Gilbert, Eric. (2015). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014.

[6] Tokunaga, Takenobu, and Iwayama Makoto. "Text categorization based on weighted inverse document frequency." *Special Interest Groups and Information Process Society of Japan SIG-IPSJ*. (1994)

[7] Zozor, Steeve, Philippe Ravier, and Olivier Buttelli. "On Lempel–Ziv complexity for multidimensional data analysis." *Physica A: Statistical Mechanics and its Applications* 345.1-2 (2005): 285-302.