AC 2012-4810: AN AUTOMATED APPROACH TO ASSESSING THE QUALITY OF CODE REVIEWS

Lakshmi Ramachandran
Dr. Edward F. Gehringer, North Carolina State University

Ed Gehringer is an Associate Professor in the departments of Computer Science and Electrical & Computer Engineering at North Carolina State University. He received his Ph.D. from Purdue University and has also taught at Carnegie Mellon University and Monash University in Australia. His research interests lie mainly in computer-supported cooperative learning.

An Automated Approach to Assessing the Quality of Project Reviews

Abstract

Peer review of code and other software documents is an integral component of a software development life cycle. In software engineering courses, peer reviewing is done by other students in the class. In order to help students improve their reviewing skills, feedback needs to be provided for the reviews written by students. The process of reviewing a review or identifying the quality of reviews can be referred to as metareviewing. Automated metareviewing ensures provision of immediate feedback to reviewers, which is likely to motivate the reviewer to improve his work and provide more useful feedback to the authors. In this work we focus especially on reviews written for software development projects. Our goal with this study is to try and identify metrics that would be best suited to study the quality of project reviews in comparison to those of reviews written for paper technical articles. Textual submissions are more likely to receive reviews that refer directly to the content of the submission as well as contain a lot of praise or criticism. Therefore metrics such as content, tone and quantity are used for such reviews. We performed experiments on review data collected from Expertiza and learnt that project reviews tend to focus more on issues related to the code and implementation of the project (they discuss testing, design aspects, code duplication) and are also not likely to contain a pronounced tone. We use natural language processing and machine learning techniques to identify and calculate the values of the new metrics. We also found that when using the new metrics to represent reviews, metareview scores were predicted with an accuracy of 84.3%, which is much higher than when just content, tone and quantity were used to predict metareview scores, which is 57.8%.

Keywords: software project reviews, quality, metrics, expertiza, metareviewing

1. Introduction

Reviews help authors to improve the quality of their work. It is therefore important to ensure that the reviews are of a good quality. Metareviewing is the process by which the quality of a review is determined. At present metareviewing is done manually, since there are no systems that help identify the quality of reviews automatically. We introduce an automated metareviewing process that helps provide quick and effective feedback on the quality of reviews.

In one of our earlier works, we introduced an automated metareviewing technique for reviews of technical articles or papers [6]. Henceforth in the paper we will refer to reviews written for technical submissions as *plain-text reviews*.

In order to be able to provide feedback on quality of a review a good understanding of the review's content is necessary. Hence, it is important to identify metrics that suitably represent reviews. For reviews written for a paper, the feedback is more likely to be summative, i.e., providing summaries of the content, or evaluative [5], i.e., offering praise or criticism to the

author's work. An example of a review written for a paper is "The example for delegation is taken from one of the references listed at the bottom of the page." This review is critical of the author's work and implies that an example on the topic "delegation" had been copied from another source. We see that the content of this review is *directly* linked to the content of the submission.

In the case of project reviews, they point out issues in the design, implementation or in the testing of the project. Such content may not be directly inferred from a description of the project, since it refers to implementation details. For example, consider the following review, "The system controller has the bulk of the functionality. It contains functions related to users, posts, replies and votes. Could have segregated these into separate sub controllers." This review talks about the *way* the project has been implemented and the distribution of functionality across modules.

Therefore, an important difference between software project reviews and plain-text reviews is that while plain-text reviews tend to reference (directly) content in the author's submission, project reviews tend to discuss the way (e.g. code syntax, design, architecture etc.) in which the project was implemented. Thus project reviews might include more references to the code or design of the project than to the content in the software requirements document. That is the content of project reviews are not restricted by the content in software documents (requirements, design etc.). This does not imply that plain-text reviews are less technical than project reviews or vice versa. These sets of reviews differ primarily in the object under discussion.

However, there may also be project reviews that are evaluative in nature, i.e., they provide positive or negative criticism of the implementation or design of the project. For instance consider the review, "Yes , good O-O style has been made use of with right MVC architecture employed." This review is praising the author's choice of object-oriented pattern. Thus although there appears to be differences in the content and the way they are written, project reviews and plain-text reviews have some similarities too.

In this paper we focus on project reviews and try to identify metrics that suitably capture their meaning. We study the differences and similarities between the two types of reviews and try to identify what metrics are more likely to work for project reviews.

This paper is organized as follows. Section 2 contains a description of our approach, which involves studying similarities and differences between the two types of reviews and ultimately identifying a set of metrics that uniquely represent project reviews. In section 3 we study the performance of each of these metrics by conducting experiments to predict metareview scores. Section 3.2. contains a discussion of our results and Section 4 concludes with a summary of our work.

2. Approach

2.1. Data collection

Peer-review data was collected from the Expertiza project at North Carolina State University [2, 3]. Expertiza is a web-based collaborative learning environment, which provides peer review and metareviewing features to instructors and students. Different types of assignments including coding projects can be hosted on Expertiza. It uses review rubrics to collect feedback from students and instructors. The feedback consists of both text-based reviews as well as numeric scores. Figure 1 shows a screenshot of a review rubric from Expertiza.

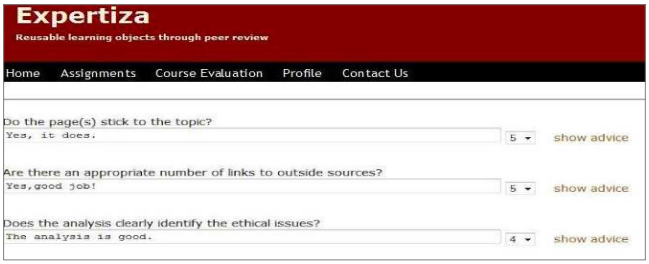

Figure 1: Screenshot of a review rubric from Expertiza, which contains textboxes for students to write out text reviews and dropdown boxes for scores.

Some examples of project review rubrics used in Expertiza are listed in Tables 1 and 2. Table 3 lists the rubric used to provide plain-text reviews for a technical article edited on the Web. As can be seen from Tables 1 and 2, the questions focus more on *how* the project was implemented than the specific functionalities of the project. Table 3, on the other hand, focuses on content (what) of the technical article. Although one question in this rubric seeks information on the page's organization (how), the responses are likely to contain references to specific content in the article itself.

Table 1: This rubric assesses the quality of the code based on its implementation, comments provided and ease of understanding.

1	Has this team avoided duplication of code?
2	Has this team incorporated design patterns into its code?
3	Has this team provided adequate comments in their code?
4	On a scale of 1 (worst) to 5 (best), how easy is it to understand the code?

Table 2: The following rubric focusses on the design aspects of the project.

1	Have the authors adequately explained the changes to be made to the system?
2	Does the design appear to be sound, following appropriate principles and using appropriate patterns?
3	Does the design appear to be as simple as possible, given the requirements?
4	Do the class diagram and/or other figures or text that clearly describe the changes to be made to the system?

Table 3: Rubric used for submissions of technical articles or papers.

1	Do the pages stick to the topic?
2	Are there an appropriate number of links to outside sources?
3	Does the analysis clearly identify the ethical issues?
4	Do the pages treat differing viewpoints fairly?
5	Is the organization of page(s) logical?
6	Do the pages identify several issues that are important in learning about the topic?

For the analysis in this paper we collected project review data from two software projects. Students were asked to evaluate the entire project based on rubrics in Tables 1 and 2, one rubric for each software project. We follow an informal, blind review process, where reviewers are given links to authors' projects, which includes code and design documents. These software artifacts are evaluated with the help of the review rubrics, and textual feedback and scores are provided to the authors. The two development projects when taken together had a set of 1427 individual text reviews.

In the next section we discuss the steps we took to identify similarities and differences between project and plain-text reviews.

2.2. Identifying similarities and differences between project reviews and plain-text reviews

In order to study factors that are likely to differentiate project reviews from plain-text reviews, we utilized a graph-based similarity-matching technique[1] to extract the most significant patterns from each of the review sets. These patterns are word phrases that capture the most frequent and semantically important *subject – verb*, *verb – object*, *subject/object – adjective* and *verb – adverb* relationships in the reviews. We use 662 plain-text reviews written for a technically worded article to identify plain-text patterns. Some patterns for each type of review are listed in Table 4.

Table 4: Frequent and semantically important patterns captured from the set of project reviews and plain-text reviews.

Patterns from project reviews	Patterns from plain-text reviews
seem have avoided - lot **code duplication** through use **helper functions** seem have been added rounding - **test suite** bit **functionality test cases** - have been added **tests cases** - have been performed use **Capybara rspec** - user admin username password	**ethics links** - are links for **Utilitarianism –** distinct **sentence definition callback** - is copied **basics MSIMD architectures** - could have been provided diagrams – explanatory

From Table 4 we can see that project reviews include discussion of *code duplication,* references to specific *helper functions,* the *test suite* and *test cases* and some language specific tools such as *capybara* and *rspec*. However in the case of patterns from plain-text reviews we see that the focus is directly on the objects in the article such as *ethics, utilitarianism, sentence definition* and *MSMID architecture.* We can also see that these reviews contain explicit praise and criticism expressed using words or phrases such as *explanatory, distinct, is copied.*

Table 5 lists some of the metrics that we consider for plain-text reviews in our earlier work [6].

For project reviews, we derive metrics that are similar to those listed in Table 5. From Table 4, we see that project reviews focus on some code specific aspects. *Summative* reviews (summaries of author's work) are less likely to occur in project reviews and therefore can be eliminated. However, there are reviews that identify problems or offer possible solutions. Hence *content* categories *problem detection* and *advisory* are retained for project reviews. In addition to that we introduce a project-review specific metric that helps identify the degree to which the review discusses the project or the code itself. This metric is referred to as *project review content.*

From Table 4, we see that project reviews do not appear to contain evaluative reviews (explicit praise or criticisms). Therefore *tone* might not be of much use while evaluating project reviews.

1 The similarity matching technique made use of Wordnet [1] to identify synonymy, hypernymy, hyponymy type of relations across word phrases.

However, *quantity* might be necessary to determine if the reviewer has provided sufficient feedback to the author. Table 6 lists some of the metrics that help represent project reviews.

Table 5: Metrics that suitably represent plain-text reviews.

Metric	Description
Content	Identifies the type of content the review contains. Our classification of content consists of three categories – **- summative** reviews provide a summary of the author's work or just a praise, **- problem detection** reviews identify problems in the author's work and **- advisory** reviews offer suggestions to the author on ways of improving the submission.
Tone	Identifies the semantic orientation of a review. Tone is classified as – **- positive** reviews have a positive semantic orientation, - **negative** reviews are those that have a negative semantic orientation and - **neutral** reviews are those that contain both positive and negative semantically oriented reviews.
Quantity	Identifies the number of tokens a review contains and helps identify the quantity of feedback provided.

Table 6: Metrics that suitably represent reviews of projects.

Metric	Description
Advisory content	Identifies the type of content the review contains. Content of project reviews can be classified as follows – **- problem detection, advisory** (similar to the categories used for plain-text reviews) or - **none**: if the reviews do not fall into either of the two previous categories
Project review content	Identifies the extent to which a review discusses the project or code.
Quantity	Identifies quantity of feedback provided.

2.3. Calculating the metrics

We use a supervised text classification technique called latent semantic analysis (LSA) to determine the content of the project reviews [4]. LSA is widely used in the field of natural-language processing to perform text classification. It produces a succinct representation of a term-document relationship matrix in a space of reduced dimensions. We then apply the cosine similarity metric to identify the document vector that is closest to a new review's vector. The closest document's class is used to identify the content category the new review would belong to.

LSA is used to calculate metrics advisory content as well as project review content. Advisory content is identified using a supervised approach, by training a model on a set of pre-tagged reviews. Project review content is identified by comparing the reviews with patterns identified from a set of training reviews. The degree of match with these patterns indicates the extent to which these reviews discuss the project. Quantity of feedback is identified by taking a count of the unique tokens in the review.

2.4. Comparing review vectors to determine metareview scores

Figure 1 shows the screenshot of a review rubric or questionnaire used by Expertiza. A complete review response includes all textual and numeric responses provided by a reviewer to an author, i.e., one completed review questionnaire. Review vectors represent complete review responses. Review vectors are therefore formed by combining (summing up) the metrics values for each of the text reviews (individual responses to review rubric questions). Categories of the metric *advisory content* are given discrete values (none – 0, problem detection – 1 and advisory – 2) so that they can be summed up easily. In general reviews that provide suggestions for improvement are more useful than those that merely identify problems or issues in the author's submission. Therefore advisory reviews are given a higher value when compared to problem detection reviews. The other two metrics *project review content* and *quantity of feedback* contain real (degree of match) and integer values respectively and hence can be summed up to obtain the final review vector representation.

Metareview scores of new reviews are identified by comparing (using cosine similarity) the review vectors of the new reviews with those of existing, metareviewed reviews. The closest metareviewed review's metareview score is used to identify the metareview score of a new review.

3. Experiments

3.1. Technique

We carry out the following experiments to identify the effectiveness of the metrics and categories we have identified for project reviews. We also compare the new set of metrics with the plain-text review metrics (listed in Table 5).

We use LSA trained a set of 636 previously annotated reviews to identify the *advisory content* of 1427 reviews.[2] We use a set of 82 semantic patterns mined from the project reviews to identify the *project review content* of the same set of reviews. Due to the availability of only limited data, we derive the patterns from the same data set. We use LSA and cosine to identify the degree to which these reviews discussed code related content.

2 The reviews were segmented as part of a pre-processing step and hence the large number of reviews.

There are a total of 213 complete review responses. This set is divided into training and testing, with 149 used for training and 64 used for testing. Metareview scores of test reviews are predicted using those of the training reviews by identifying the closest training review. Metareview scores are given on a Likert scale with values from 1 to 5, where 1 is the lowest and 5 is the highest. Metareview scores of a complete review is determined by taking the average of metareview scores awarded to each (metareview) rubric[3] question.

3.2. Results and Analysis

We predict accuracy by determining if the predicted metascores are within 1 unit of the actual metareview scores, i.e., (**abs (predicted metareview score – actual metareview score) <=** 1). Since averaged metareview scores take on real values, looking for an exact match between the predicted and actual values might be a bit too constraining. Therefore we use the above-stated condition to study the extent to which predicted metareview scores agree with those given by metareviewers.

Table 7 lists the accuracy values of predicting metareview scores for project reviews using the identified set of metrics. The accuracy we get when using the new set of metrics is 84.3%, i.e., the set of predicted scores that satisfied the above-mentioned condition. The accuracy of predicting metareview scores using the plain-text reviews' metrics is 57.8%. Thus we see that for the current training and testing data, the system is able to predict metareview scores with a higher accuracy while using the new set of metrics. Some examples of correctly classified project reviews are listed in Table 8.

Table 7: Accuracies of predicting metareview scores for project reviews using different sets of metrics.

Metrics used	**Accuracy of predicting metareview scores**
Content, tone and quantity	57.8%
Advisory content, project review content and quantity	**84.3%**

We also notice that for 28.1% of the test reviews (while using the plain-text reviews' metrics), metareviewers had given reviewers high scores (>> 1) although no textual feedback was provided. Metareviewers tend to be quite generous and award reviewers very high scores. Metareviewers must be provided with a suitable rubric that will guide them better during the metareview process. This could ensure more accurate metareview scores for our experiments in the future.

3 Metareview rubrics are similar to review rubrics and allow metareviewers to provide textual and numeric feedback, but for our experiments we focus only on the numeric scores.

Table 8: Examples of project reviews that were correctly classified using the new set of metrics.

1	No design patterns involved in this code. I will however require clarification from author regarding the same . < < UPDATE > > The project mainly dealt with performance analysis. So no design patterns were involved. Some more comments could have been provided < < UPDATE > > Comments are added as requested.
2	The testcases cover the functionality required. There are different testcases. However, I still would require a clarification from the authors on which tests to look into and a readme file would be great. Strongly advice you to write a README file indicating what is done and where to find the required code and also how to run the tests . I did a bit of digging myself still could not figure out which tests were performed by you and how to review the same . < < UPDATE > > ReadMe provided with detailed description.
3	Have u deployed the code on VCL to run the tests ? I currently have my project hosted on VCL and hence can't take another session. Can u provide me a VCL session to verify the testcases?
4	It seems like the design patterns that were already followed by expertiza has been retained and also introduced some. Yes. Modules have been made. Some functionality moved to other models. Testing done.

4. Conclusions

In this paper we have identified a new set of metrics to determine the quality of project reviews. We found that project reviews discuss content (e.g. test cases, specific functions, software tools etc.) that may not refer directly to the project description or requirements, in contrast to plain-text reviews, whose contents are often directly related to the technical article or paper's text. We also notice that project reviews do not contain a lot of praise or criticism like the plain-text reviews do. We use the new metrics *advisory content, project review content* and *quantity* to predict metareview scores of project reviews. We found that project reviews produce higher accuracy values with the new metrics than with metrics *content, tone* and *quantity* used by plain-text reviews.

References

[1] Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

[2] E. F. Gehringer, L. M. Ehresman, and W. P. Conger, S.G., "Reusable learning objects through peer review: The Expertiza approach," in *Innovate: Journal of On- line Education,* 2007.

[3] E. F. Gehringer, "Expertiza: information management for collaborative learning," in *In Monitoring and Assessment in Online Collaborative Environments: Emergent Computational Technologies for E-Learning Support.* IGI Global Press, 2009.

[4] F. P. W. Landauer, T. K. and D. Laham. An introduction to latent semantic analysis. In *Discourse Processes. Special Issue on Quantitative Approaches to Semantic Knowledge Representations,* 259–284. Volume 25. 1998.

[5] T. A. S. Pardo, L. H. M. Rino and Maria das Graças Volpe Nunes, "Extractive summarization: How to identify the gist of a text," in *Proceedings of the 1st International Information Technology Symposium - I2TS*, Florianpolis-SC, Brazil, October 1-5 2002, pp. 1–6.

[6] Lakshmi Ramachandran and Edward F. Gehringer. 2011. Automated assessment of review quality using latent semantic analysis. *11th IEEE International Conference on Advanced Learning Technologies.* 136-138 July.