Report on Evaluating Code Quality Through Code Reviews

Anusha Mogili
19970626-8167
moan17@student.bth.se

Vadrevu Krithi Sameera
19961031-6086
vakr17@student.bth.se

Kanneganti Joshika
19970429-1781
kajo17@student.bth.se

Manoj Kumar Pallapu
19970919-2018
pama17@student.bth.se

Abstract

This paper presents evaluation of code quality through code review of 'Jazzy - Java Spell Check API' an open source software. To analyse code quality, code review is performed manually and by using static code analysis tools Codacy and Codebeat. A review checklist is created using review themes like formatting, architecture, testing etc. The other part of paper presents detail study of tool outcomes and also comparison of manual evaluation and tool evaluation of the code quality.

Keywords: Code quality, Code review, Static code analysis tools.

1. Introduction

In today's world, Code quality plays a vital role in software development. Evaluation of Code quality helps to improve reliability and consistency of code. Code review activity improves readability, maintainability, understandability and also identify bug frequency which assures quality. Static code analysis tools facilitate code review process which helps in identifying majority of the vulnerabilities. They provide various measures, different patterns, inspect overall code coverage to increase security, maintainability, helps to avoid duplications, and also high complexity issues. To check the outcomes of the code review tool, manual reviews come in hand for cross-checking. Automated tools mainly focus on coding standards and formats, whereas manual code review process helps to identify design level issues.

In this paper, in order to evaluate the code quality of 'Jazzy - Java Spell Check API' which is an open source java project based on most of the algorithms that spell has [7]. we have taken a module *'com.swabunga.spell.event'* from the source code which consists of 17 classes. Code review conducted using tools and also manually using checklists is discussed in section 2. The tool values are compared, and their outcomes are discussed in section 3. The Quality evaluation performed manually and by using tools are compared in section 4.

2. Review Themes

Table 1. Review checklist

S.no	Themes	Question	Aspect Covered
1.	Code formatting	a) Are there more than required number of logical operators in single condition? [6]	✓
		b) Will unused variables affect understandability? [4]	✓
		c) Do we need to remove reassigning parameters? [5]	✓
		d) Does nested loop lead to a complicated design? [5]	✓
		e) Does duplicate code create uncertainty in code? [2][3]	✓
2.	Coding standards	a) Is having unused and uncommented methods reduce readability? [4]	✓
		b) Does Commented Lines of Code bloat the program? [5]	✓
		c) Are the declared variables and methods following naming conventions? [4]	✓
		d) Do methods follow coding standards? [4]	✓
3.	Architecture/Design	a) Does TLD (Test Last Development) mean real improvement in code? [1]	
		b) Do multiple responsibilities given to single class effect code structure? [2]	✓
		c) Does the code follow modular design technique? [2]	
4.	Testing	a) Do tests inspect all the failure conditions? [1]	
		b) Will tests cover all paths of production code? [1]	
5.	Non-Functional	a) Is the code thread safe? [5]	✓
		b) Are invalid data inputs handled? [5]	

Review Questions/defect/aspect

Architectural review theme
Changes in internal structure of the code needs deep knowledge to figure out functionalities of the code.

Code Formatting review theme
It is a way of improving code by reducing it and optimising code to meet quality requirements. It need not be always reduced as sometimes adding code improves code readability.

Testing review theme
Check if tests are framed for all production code paths. Do not proceed with reviewing process without understanding goals of the production code and spend considerable amount of time on it.

Coding standards theme
These standards when violated does not create any run time errors but the purpose of making code 'readable to all' is affected.

Non-functional theme
Enhancements in non-functional requirements like performance, security, maintainability, readability etc have a trade-off. Based on the need, we prioritize the specific non-functional requirement.

3. Results

Comparison and Reflection of tools

Codacy and Codebeat are effective code review tools with their own defined metrics, and also helps in understanding code structure and to check coding standards by evaluating possible vulnerabilities.

Codacy mainly evaluates three attributes such as Size, Structure, Complexity, and also measure of Duplication. For each of these attributes, various metrics are provided to analyze code quality, security issues and time taken to fix each bug. Codacy is known for producing helpful advice in fixing the issues. We can configure the threshold values and also select useful patterns for necessary files. Metric values can be generated for a particular class and also overall issues can be extracted for every package.

Codebeat help us in evaluating Complexity, Duplication, Size and list of code issues with code coverage feature. This does not give security issues, for measuring the code quality. It will give us the quality of the software in the form of scale rating (A-F). we can't configure the threshold value.
The values extracted from the two tools consists of different values with respect to different aspects, such as:

→ **Complexity** - The complexity values generated by both the tools differ by their definition such as, in Codacy n-path complexity is measured whereas in Codebeat cyclomatic complexity metric value is extracted by checking every branch statements.

→ **Size** - Codebeat measures the source lines of code and Codacy measures the source Lines of code and Commented lines of code to give total lines of code.

→ **Methods -** Codebeat shows the total number of methods in the class whereas in the Codacy it evaluates the method name, method length and some other patterns if present. But it doesn't show the true value of total number of methods in the code.

→ **Duplicate -** In Codebeat, it shows the similar count of repeated code in the same class i.e.

→ duplicated code, whereas in the Codacy the number of clones i.e., repeated branch of code and duplicate lines of code is shown for every class.

→ **Security** - Codebeat doesn't show the security issues for the source code whereas in the Codacy gives the complete analysis of the security issues in the form of values.
The values given by the tools are true values as we have manually evaluated the code and compared them with the tool values and they were similar. Except for Number of methods in the Codacy tool.

SpellChecker.java and *Java.WordFinder.java* has high complexity as its value is above threshold. As the complexity increases, maintainability reduces.
With the increase in number of methods per class in *Spellchecker.java*, the readability decreases too. This reduces maintainability.
As the lines of code for a class increases, the time spent on analysing it will increase. Hence maintainability of *Spellchecker.java* reduces.

As duplicate code increases, any modification in one block requires other similar blocks to be changed. As modifiability increases maintainability reduces.

Table 2. Measures reported by Codacy

	Complexity	Duplication	Methods	Lines of code
AbstractWordFinder.java	82	0	19	157
AbstractWordTokenizer.java	12	0	10	35
BasicSpellCheckEvent.java	15	0	11	50
DefaultWordFinder.java	27	0	4	49
DocumentWordTokenizer.java	77	1	12	138
FileWordTokenizer.java	13	0	4	25
JavaWordFinder.java	32	0	4	62
SpellChecker.java	148	0	27	238
SpellCheckerEvent.java	0	0	10	10
SpellCheckerListener.java	0	0	1	1
StringWordTokenizer.java	3	0	5	16
TeXWordFinder.java	47	0	6	77
Word.java	12	0	11	39
WordFinder.java	0	0	7	7
WordNotFoundException.java	0	0	2	6
WordTokenizer.java	0	0	8	8
XMLWordFinder.java	18	1	2	37

Table 3. Measures reported by Codebeat

	Lines of code	Source lines of code	Commented lines of code	Complexity	Number of Clones	Duplicated lines of code
AbstractWordFinder.java	392	166	161	8	0	0
AbstractWordTokenizer.java	164	43	91	3	0	0
BasicSpellCheckEvent.java	147	62	66	4	0	0
DefaultWordFinder.java	116	53	48	11	0	0
DocumentWordTokenizer.java	625	157	88	9	0	0
FileWordTokenizer.java	92	32	46	3	0	0
JavaWordFinder.java	139	66	47	17	0	0
SpellChecker.java	574	261	234	18	0	0
SpellCheckerEvent.java	102	21	66	Nil	0	0
SpellCheckerListener.java	36	5	25	Nil	0	0
StringWordTokenizer.java	91	19	58	1	0	0
TeXWordFinder.java	188	87	73	13	1	30
Word.java	141	45	72	1	0	0
WordFinder.java	102	10	78	Nil	0	0
WordNotFoundException.java	44	9	28	1	0	0
WordTokenizer.java	105	11	81	Nil	0	0
XMLWordFinder.java	97	40	39	13	1	30

1. Evaluation of Code Quality

4.1 Reflections

→ Codebeat and Codacy cover the breadth of the code like "condition too high" which occurs in more than one class whereas manual reviewing covers depth.

→ These tools are more reliable when it comes to review as we tend to make mistakes during repetition of issues.

→ There is less chance of occurrence of false positives and false negatives when evaluation is performed manually compared to tools.

→ During manual evaluation metrics are not calculated. Hence, uncertainty resides in the review we performed. It is also more likely to skip over aspects.

→ We did not look for accidental errors while manually evaluating code. Moreover, design level issues can be better analysed using manual review.

→ We spend more time and effort for conducting manual review. All the tool need is to setup the environment to perform review.

→ Review conducted with tools is much faster compared to manual.

→ The tools Codebeat and Codacy, display the block of code which has issues and needs to be fixed.

4.2 Issues Identified

→ **Issue:** Presence of Identical code in multiple locations.

Impact: If modifications are done in one place the identical blocks need to be changed too.

Affected Classes: DocumentWordTokenizer.java, XMLWordFinder.java.

Quality Aspect: Complexity.

Solution: Placing the identical code block in a method and calling it the wherever required.

Possible Further issues: But sometimes, if the number of occurrence of duplication is less than its better to retain it.

Reasons: It would make code more readable and maintainable.

→ **Issue:** Avoid Unused Local Variables

Impact: Presence of unused variables creates confusion and leads to difficulty in analysing the source code.

Affected Classes: JavaWordFinder.java, FileWordTokenizer.java.

Quality aspect: Complexity, Understandability.

Solution: Removal of unused variables in methods and classes helps in clearly understanding source code.

Possible Further issues: Unused variables may lead to confusion in referring to the actual property.

Reasons: Unused variables with same name may create confusion for referring to the proper object or method but they help while debugging (used for inspecting return values).

→ **Issue:** Commented Lines of code is bad.

Impact: Due to these commented lines of code it will cause the difficulty for a developer to analyze the code.

Affected Classes: TextWordFinder.java.

Quality aspect: Complexity, analyzability

Solution: Removing the commented lines of code in source code will improve analysability of the code in turn quality improves.

Possible Further issues: commented lines of code is a dead code which people may fear to remove as they spend time on it. Hence, compiler will be able to optimize it, even with its presence.

Reasons: Code will be up to date without reducing performance.

→ **Issue:** Function Too Long.

Impact: increases the complexity

Affected Classes: SpellChecker.java, XMLWordFinder.java, JavaWordFinder.java, TexWordFinder.java.

Quality aspect: Performance

Solution: Break the function into smaller functions in a way that each operation/functionality has its own function.

Possible Further issues: Many arguments are declared inside the function.

Reasons: To identify which operation uses which variable is confusing at times.

→ **Issue:** Assignment Branch Condition Too High.
Impact: If many if conditions are written it takes time to analyse the purpose of code.
Affected Classes: SpellChecker.java, XMLWordFinder.java, JavaWordFinder.java, TexWordFinder.java, AbstractWordFinder.java, DocumentWordTokenizer.java, DefaultWordFinder.java, FileWordTokenizer.java.
Quality aspects: Readability and Understandability.
Solution: Define a new method and reduce the nested if statements with an if condition inside a for loop.
Possible Further issues: It is difficult for another programmer to understand the code.
Reason: Code must be understood by anyone without need of the programmer**.**

References

[1] D. Spadini, M. Aniche, M.-A. Storey, M. Bruntink, y A. Bacchelli, "When Testing Meets Code Review: Why and How Developers Review Tests", en *Proceedings of the 40th International Conference on Software Engineering*, New York, NY, USA, 2018, pp. 677–687.

[2] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, y P. Avgeriou, "Assessing Code Smell Interest Probability: A Case Study", en *Proceedings of the XP2017 Scientific Workshops*, New York, NY, USA, 2017, pp. 5:1–5:8.

[3] "Technical debt as an external software attribute". [En línea]. Disponible en: https://dl-acm-org.miman.bib.bth.se/citation.cfm?id=3194168. [Consultado: 08-dic-2018].

[4] S. Krusche, M. Berisha, y B. Bruegge, "Teaching Code Review Management Using Branch Based Workflows", en *Proceedings of the 38th International Conference on Software Engineering Companion*, New York, NY, USA, 2016, pp. 384–393.

[5] "Code Review Checklist – To Perform Effective Code Reviews", Evoke Technologies Blog, 31-ago-2015.

[6] Y. Ueda, A. Ihara, T. Ishio, T. Hirao, y K. Matsumoto, "How are IF-Conditional Statements Fixed Through Peer CodeReview?", *IEICE Trans. Inf. Syst.*, vol. E101.D, núm. 11, pp. 2720–2729, nov. 2018.

[7] "Jazzy - Java Spell Check API - Browse Files at SourceForge.net." [Online]. Available: https://sourceforge.net/projects/jazzy/files/.[Accessed: 03-Dec-2018].

[8] "The Pros and Cons of Four Kinds of Code Reviews", *CMCrossroads*. [En línea]. Disponible en: https://www.cmcrossroads.com/article/pros-and-cons-four-kinds-code-reviews. [Consultado: 03-dic-2018].

[9] S. Vonnegut, "5 Best Practices for the Perfect Secure Code Review", *Checkmarx*, 05-feb-2016.