Code Quality: Examining the Efficacy of Automated Tools

Emergent Research Forum Paper

Sara Hooshangi
The George Washington University
shoosh@gwu.edu

Subhasish Dasgupta
The George Washington University
dasgupta@gwu.edu

Abstract

A number of software tools have been designed to determine code quality. These automated tools can examine a computer program and provide a score based on a designed set of software metrics. In this exploratory study, we examined the quality of code written by students in an introductory Python programming course on one programming assignment. Each student submission was evaluated using an automated tool, Pylint, a code analyzer widely used by the Python community. The instructor also graded these submissions using predefined rubrics that evaluated code logic, syntax and style. We compared the two code quality scores. We found that Pylint does a good job of identifying errors, and formatting issues. But, the Pylint scores were lower than those provided manually by instructors.

Keywords

Code quality, Python, Automated grading tools

Introduction

Determination of code quality is an important area of research in computer science and information systems. A number of metrics have been used over the years to determine code quality. Metrics of code quality have been used widely to evaluate the integrity and functionality of computer code and software in organizations, and in some open source platforms such as Github (Ray, Posnett, Filkov, & Devanbu, 2014). In a few cases code quality metrics have been used in educational settings to determine the quality of student code (Altadmri & Brown, 2015; Athanasiou, Nugroho, Visser, & Zaidman, 2014; Breuker, Derriks, & Brunekreef, 2011; Kasto & Whalley, 2013; McCracken et al., 2001). In addition, a number of software tools have been designed to determine code quality. These automated tools can examine a computer program and provide a score based on a designed set of software metrics. Such automated tools are useful for a couple of reasons. First, automated tools provide a standardized methodology to determine quality by evaluating code based on a set of pre-defined metrics. Metrics fit well with rubrics that instructors use in classes. Second, automated tools can provide consistency in measurement, and reduce the chance of human error. Lastly, determining quality of code is a manual and labor-intensive process. Automated tools can help reduce the time required when there is a large amount of code to be examined.

In this exploratory study, we examined the quality of code written by students in an introductory Python programming course on a programming assignment. Each student submission was evaluated using an automated tool, Pylint (https://www.pylint.org). Pylint is a code analyzer widely used by the Python community. The instructor also graded these submissions using predefined rubrics that evaluated code logic, syntax and style. We then compared code quality scores provided by Pylint with grades provided by the instructor. The objective of this study was to examine the efficacy of automated tools such as Pylint to determine code quality of student assignments.

The paper is organized as follows. In the next section, we provide a theoretical framework. This is followed by research methodology, results, and conclusion.

Theoretical Framework

The ISO/IEC 9126-1 standard for software product quality identifies three parts of quality: internal quality, external quality, and quality in use. This standard specifies metrics for each of the three parts of software product quality. "Internal metrics can be applied to a non-executable software product (such as a specification or source code) during designing and coding. External metrics use measures of a software product derived from measures of the behavior of the system of which it is a part, by testing, operating and observing the executable software or system. Quality in use metrics measure the extent to which a product meets the needs of specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in a specified context of use. Evaluating quality in use validates software product quality in specific user-task scenarios." (ISO/IEC, 2001, p. 15). In this research, we focus on the internal metrics of software product quality, which applies to source code – we refer to this as code quality.

Methodology

Data Collection

This study focused on the analysis of a Python assignment that was given to students in an introductory programming course. The students' task was to write three functions to calculate the areas of a predefined list of shapes and find the largest and smallest areas. The predefined list located in the main body of the program, contained six tuples where each tuple contained a string to represent the shape (either a rectangle or a triangle) and the associated dimensions for that shape in forms of two float numbers. Students were asked to write a separate function for each area calculation and to also include a parser function, to determine which area function was chosen for the calculation. This assignment was given mid semester when students were familiar with basic Python syntax and had work with function constructs, if statements, for loops and variable assignments. Forty-four undergraduate students completed this assignment where 52% (23 students) were male and 48%(21 students) were female in this sample. Students' age ranged from 22-40.

Data Analysis and Results

Pylint Score
Pylint is an automated tool that can be used to assess the quality of a python code based on a weighted linear calculation that takes into account number of errors, warnings, refractors and conventions found during code analysis. Typically, warning and convention flags are raised when the code does not follow standard Python convention or style guides or there is an issue with variable assignments. For example, missing docstring for a function or a module or bad whitespace are considered convention flags. A warning might be related to the use of a keyboard as a variable name or having unused variables in the code.

A typical Pylint output is a basic terminal output that includes all the warnings, convention issues and error messages, basic statistics about the code (such as number of modules, classes, and methods) and raw metrics related to the number of lines of actual code, docstring, comments and empty space within the code. A total rating score is provided at the end of the output as an overall measure of the code quality. Pylint uses the formula below to calculate the rating score, where "statement" is the number of lines of code analyzed by Pylint.

$$\text{Pylint Score} = 10.0 - (5 * \text{error} + \text{warning} + \text{refactor} + \text{convention}) / \text{statement}) * 10)$$

Instructor Score
In addition to the automated score that was used to evaluate students' code in this project, we also used an instructor-based score to measure code quality. The instructor went through each student submission and assigned a score between 0-5 for each of the six categories shown in Table 1. Each rubric measure category had a weight associated with it. The weight corresponded to the importance of that particular metric in the instructor's assessment. For example, the measure called "Quality of Run" which determined whether or not a piece of code run with or without syntax and runtime errors was given a weight of 2.

Metrics such as extent of docstring usage or the number of lines of code in the program, were seen as less important and were given a weight of 1. The overall score was calculated using the following formula:

$$\text{Instructor score} = \left(\sum_6 Rubric\_score * weight\right)/5$$

Rubric Measure (Assigned a score between 0-5)		**Weight**
# of Functions		3
Quality of Comments		1
Good Use of Variable Definition and Name		2
Length of Line		1
Use of Docstring		1
Quality of Run (Syntax/Runtime Error Check)		2
	Total weight	10

Table 1: Rubric measures used by the instructor to evaluate the quality of student code. Each measure was assigned a value between 0-5, where 5 was the highest rating and 0 the lowest rating. Measures were weighted based on their importance in the intsurctor's assessment.

Table 2 shows the result of the instructor grading:

Rubric metric used by the instructor to grade	
Avg. Instructor Score	8.22
Avg. Quality of Run (Syntax Error Check)	4.63
Avg. Good Use of Variable Definition & Name	3.99
Avg. Length of Line	3.69
Avg. Use of Docstring	1.72
Avg. Quality of Comments	3.52
Avg. # Of Lines Of Code	40.61

Std of metrics used by the instructor to grade	
Std. dev. of Instructor Score	1.12
Std. dev. of Quality of Run (Syntax Error Che..	1.08
Std. dev. of Good Use of Variable Definition ..	1.53
Std. dev. of Length of Line	1.33
Std. dev. of Use of Docstring	1.63
Std. dev. of Quality of Comments	2.20
Std. dev. of # Of Lines Of Code	6.55

Table 2: Basic statistics on the instructor graded measures.

Our research question was to determine whether an automated code quality analyzer tool such as Pylint could provide code quality measures that are close to those provided manually by instructors. Table 3 shows a basic side-by-side comparison of the scores obtained through each method. Figure 1 represents the scores of each student assignment as a function of number of lines of the code.

Instructor versus Pylint Scores	
Avg. Instructor Score	8.22
Min. Instructor Score	4.80
Max. Instructor Score	10.00
Avg. Pylint Score	6.46
Min. Pylint Score	-0.88
Max. Pylint Score	9.33

Table 3: Instructor versus Pylint Score Comparison

We performed a t-test using the sample data to examine the difference between the Pylint and instructor mean scores. The t-test technique we used was a two-sample t-test assuming unequal variances.
Our null hypothesis is: H_o: $\mu_{pylint} = \mu_{instructor}$.

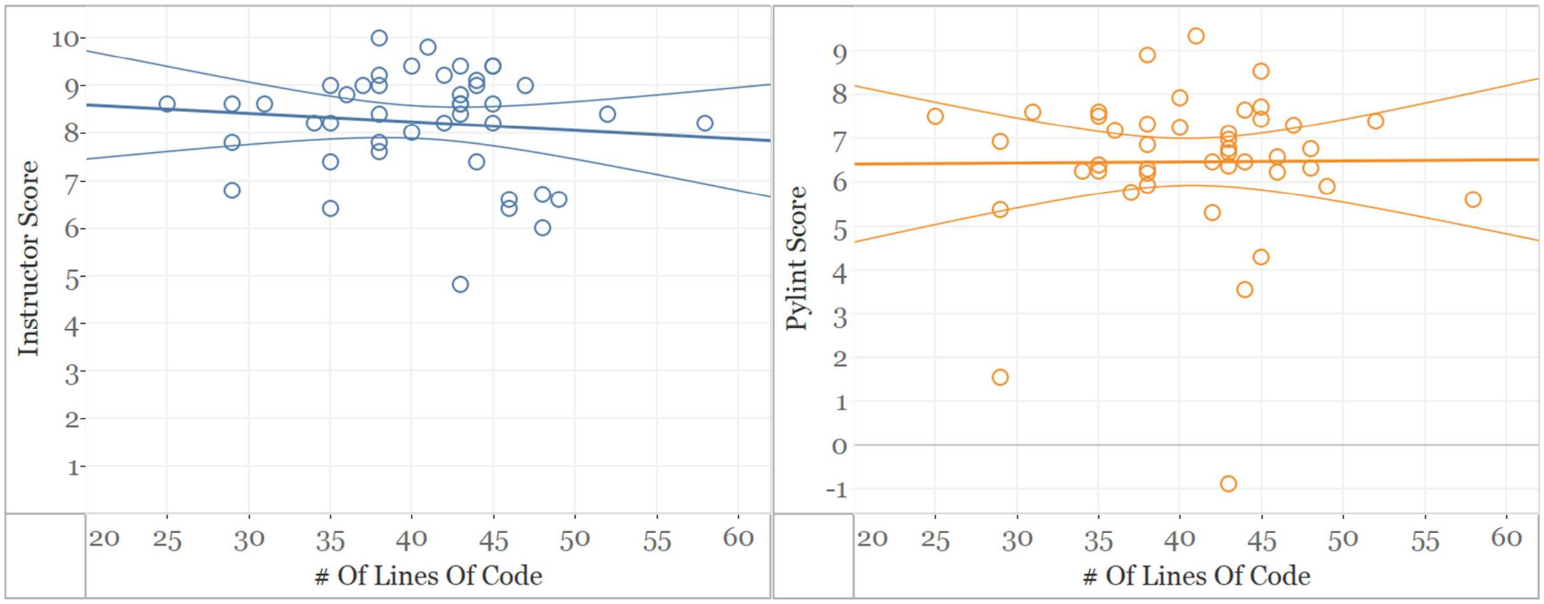

Figure 1: The score for each student as a function of number of lines of code in the submission. The left panel represents instructor's manual scoring based on the Table 1 measures, and the right panel represents the overall score produced by Pylint when each submission is analyzed by Pylint.

Table 4 below shows the results of our t-test. Based on the hypothesis, this was a two-tailed t-test. The results indicate that there is a significant difference between the Pylint and Instructor score means at the 0.001 level. Instructor code quality scores were higher than Pylint scores for graded assignment. We attempt to explain the reasons for this difference in the discussion section below.

	Instructor_Score	Pylint_score
Mean	8.218	6.462
Variance	1.246	3.0213
Observations	44	44
Hypothesized Mean Difference	0	
df	73	
t Statistic	5.64	
P(T<=t) two-tail	3.033×10^{-7}*	
t Critical two-tail	1.993	

*p-value < 0.001

Table 4: t-Test Results: Comparison of Instructor and Pylint Scores (2A)

Discussion

Our results indicate that instructor code quality scores were higher than scores calculated by our automated tool, Pylint. The difference was statistically significant. We identified a couple of reasons that could have contributed to the difference in scores. Pylint puts greater emphasis on formatting issues such as empty lines, bad whitespace, and missing docstring than an instructor. Here are a few specific instances that Pylint scored poorly. Some the students copied and pasted the entire assignment and used that as the module docstring. That results in too long sentences, which Pylint takes scores negatively.

Some students used the keywords max and min as variables names and that also didn't score well with Pylint. Majority of students didn't include a separate docstring for their functions, however their codes run without any syntax or runtime errors.

Secondly, the difference in instructor and Pylint scores could be due to differences in weights given to components of the code quality score. Both the instructor as well as Pylint use standard metrics for code quality score. Pylint tends to give all the components of code quality the same weight (it does control for the length of the program). Instructors, on the other hand, provide different weights to individual code metrics based on the importance of that component. This could contribute to the difference in instructor and Pylint scores. Although instructor and Pylint scores were different, Pylint provided consistent scores for all programs in this study. In summary, Pylint code quality components and scores should reflect an instructor's grading policies.

Conclusion

In this exploratory study, we examined the efficacy of an automated tool in determining code quality. The automated tool, Pylint, provided us with detailed description of errors based on its own metrics for code quality. Some of its measures, e.g. bad whitespace, and number of blank lines, did not reflect the grading criteria put forward by the instructor. Although we found the automated tool to be useful, we believe that additional studies are needed. Our research is a preliminary study and has its limitations. We conducted this research with only one tool, Pylint. Our study should be extended to include two or more code analyzers so that we can evaluate the efficacy of other automated tools. Moreover, our sample size of 44 students was small, and larger sample sizes should be considered. Finally, our research was conducted in an educational institution using novice programmers. This should be extended to a real-life work setting that includes professional or advanced programmers. We also believe that tools such as Pylint can help students become better coders by providing them with an easy mechanism to check the quality of their codes before assignment submission. In summary, we believe our research is a step in the right direction.

REFERENCES

Altadmri, A., & Brown, N. C. C. (2015). 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data (pp. 522–527). ACM Press. https://doi.org/10.1145/2676723.2677258

Athanasiou, D., Nugroho, A., Visser, J., & Zaidman, A. (2014). Test Code Quality and Its Relation to Issue Handling Performance. *IEEE Transactions on Software Engineering*, *40*(11), 1100–1125. https://doi.org/10.1109/TSE.2014.2342227

Breuker, D., Derriks, J., & Brunekreef, J. (2011). Measuring static quality of student code. In *Proceedings of the 16th annual joint conference on innovation and technology in computer science education* (pp. 13–17). ACM. https://doi.org/10.1145/1999747.1999754

ISO/IEC. (2001). Information technology - Software product quality - Part 1: Quality model.

Kasto, N., & Whalley, J. (2013). Measuring the difficulty of code comprehension tasks using software metrics. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136* (pp. 59–65). Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=2667206

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., … Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125–180.

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014). A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on foundations of software engineering* (pp. 155–165). ACM. https://doi.org/10.1145/2635868.2635922