Applying Code Quality Detection in Online Programming Judge

Xiao Liu
Department of Electrical and Computer Engineering
Pusan National University
Republic of Korea
+82-51-510-3939
liuxiao@pusan.ac.kr

Gyun Woo
Department of Electrical and Computer Engineering
Pusan National University
Republic of Korea
+82-51-510-3518
woogyun@pusan.ac.kr

ABSTRACT
This article presents an enhanced programming online judge system that not only evaluates the correctness of the submitted program code but also detects its code quality. Both results of the correctness and quality detection are responded through web once the compilation, the execution and the quality detection of the submitted source code have been finished. We take advantage of SonarQube to provide code quality detection in our online judge system named neoESPA. Comparing with other online judges, our proposed work has significant advantages in helping both the instructor to discover the weaknesses in the lecture and the students to locate their mistakes efficiently.

CCS Concepts
• Applied computing→Education→Computer-managed instruction

Keywords
Code quality detection; Programming code assessment; Online judge

1. INTRODUCTION
It is hard to find an automated system that not only checks the correctness of the output but also detects its code quality. In a programming course, the students need lots of practice in writing codes to improve their programming skills [1]. Meanwhile, the instructor needs to find out the learning progress by assessing the code assignments submitted by the students [2]. It is a burdensome work that assessing each coding assignment for all the students, not to mention there is a growing number of non-CS majoring students who start learning to program in colleges [3].

There are several online programming judges (OPJs) have been proposed [4]. Some of them focus on the particular programming languages such as C [5] and Java [6], some of them are used for the programming contests [7], and there are also some systems used to analyze the relationship between successful code submissions and academic scores [8].

ICIIT 2020, February 19–22, 2020, Hanoi, Viet Nam

ACM ISBN 978-1-4503-7659-4/19/07…$15.00
DOI: https://doi.org/10.1145/3385209.3385226

One of the OPJs called neoESPA has served a few years in our department for programming assignment evaluation. NeoESPA is developed from 2014 to 2015 and is a successor of an older OPJ named ESPA. It is able to handle the assessment of the code for the programming assignments written in C/C++, Java, and Python [12].

However, none of the existing systems is able to detect the code quality of the submitted programs. Since the industries require the quality of the software as much as its correctness, it is important to enhance the education of code quality management when the students learning the programming languages [9].

The code quality consists of the concepts of capability, usability, performance, reliability, and maintainability [10]. The concepts affect the programs in their efficiency, vulnerability, and security. Since the code quality depends on how the programming code is written, it is more appropriate that keeping a good programming style when learning a new programming language. In addition, it will be more efficient to fix the issues for the students to obtain their code quality rapidly rather than learning.

There are several tools to detect the quality of the software [11]. The open-source projects such as Sparse, SonarQube, and Splint provide static analysis on the target programs to check the coding mistakes. There are also several commercial software such as TestLodge, IBM Rational Quality Manager, and VerionOne that provide practical features on quality management.

This paper presents an enhanced OPJ with code quality detection to evaluate the student's code for the programming assignment more comprehensively. We apply the functions of SonarQube as a new module in neoESPA to implement code quality detection on students' codes. The enhanced neoESPA is able to not only assess the correctness of the code for the programming assignment but also detect its quality.

In order to verify the practicality of our proposed system, we collect two thousand codes to analyze their code quality in the enhanced neoESPA. The codes are written in Python and were submitted by the undergraduate students in our department during past semesters. We aim to analyze the code quality by using our proposed system to discover the most frequent mistakes made by the students. We believe that the experiment result can help the instructor to understand the weak parts of programming education. The result also can be used as suggestions for the students to locate and fix their coding mistakes more efficiently.

The rest of this paper is organized as follows. Section 2 presents the background knowledge such as similar work, the mechanisms

of code quality detection in SonarQube, and neoESPA as an OPJ system. Section 3 describes the development of enhanced neoESPA. Section 4 illustrates the experiment of using our proposed system. Section 5 discusses the meanings of the experiment result and limitations. Section 6 concludes.

2. BACKGROUND

In this section, two related studies and the concept of code quality detection in SonarQube is introduced. The procedure of a common OPJ is also described in this section.

2.1 Similar Work

In order to summarize the quality issues that occurred in students' codes, Stegeman et al. [16] proposed a study based on generic rubrics to get feedback on code quality in programming courses. They categorized ten models of code quality by referring to professional software literature and interviews with teachers. The models used for checking quality issues such as complexity, expression collapsibility, code duplication, and coupling level. The feedback can be used to analyze the lecturing of programming courses. The study collects the professional rules to detect the code quality fairly. However, it requires the human grader to evaluate the code quality of programming assignments which could be a burdensome work.

Altadmri and Brown [17] ran a black box database to test massive students' Java codes to find out the common mistakes the students made. The study analyzed the dataset by using pre-categorized eighteen mistake labels and recorded the frequencies, distinct users, median time to fix, and error types of the mistakes. Based on the investigated result, they found the most frequent mistakes that students made and counted the time of the students fixing them. The result can be used for educational purposes such as informing course and textbook design. The issues found in the study are actually from the students' source codes which are more representative of the real problems of students. However, since students are learning more than one programming language such as Java in the university, the programs in other languages should also be measured for code quality.

2.2 SonarQube

SonarQube is a platform that provides functions to manage the quality of software [13]. SonarQube consists of two parts: a server and a scanner. The SonarQube server provides an environment to manage and execute the data and functions. The SonarQube scanner runs on the server as one of the functions to detect code quality. SonarQube also provides a web front-end and a command-line interface (CLI) for users. Once the SonarQube server started running, the scanner can be executed through either CLI or web.

The quality detection features of SonarQube rely on the principles which summarized based on programming regulations and de facto patterns used to standardize the programming style. The principles are regulated into rules in SonarQube, used to measure whether a program code is developed well or not. The rules cover several parts of code quality such as efficiency, robustness, and security [14].

SonarQube also allows users to add their own rules to detect particular programming issues. According to the criticality, a custom rule can be assigned to four levels which are bugs, vulnerabilities, security hotspots, and code smells, respectively. Also, it provides two methods to add custom rules: XPath and Java Plugin API. The XPath can be used to generate Python coding rules; the Java Plugin API is a natural way to create Java programming rules.

The violations of rules detected by SonarQube can be classified by severities. The most significant violations are classified as bugs that indicate some mistakes that may lead to critical execution problems. The secondly significant violations are classified as vulnerabilities that indicate some mistakes that may lead to security problems. The most non-significant violations are classified as code smells that indicate the program may run correctly but there are potential mistakes that affect the execution.

The violations of rules are called issues in SonarQube. There are several types of issues such as Convention, Unused, MISRA, Brain-overload, Suspicious, Design, Clumsy, CERT, CWE, and Pitfall. The issues tagged with the same type can be solved with similar approaches. In addition, an issue can be tagged with more than one type. For those readers who may be interested in the issue types, SonarQube provides an introduction of the built-in rule tags [15].

2.3 NeoESPA as an OPJ

There are various OPJs serve at different institutions. We developed neoESPA to archive the evaluation for the correctness of students' programming codes through multiple test cases. Similar to other OPJs, neoESPA runs at a remote server and provides an automatic assessment of programming codes through the web.

The student can submit his/her source code to the compiler of neoESPA through the web. Once the compilation is completed, an executable program is generated based on the code. The program is then executed by the executor with several predefined criteria inputs. After the execution, the output of the program is considered as an executed result and compared with the predefined criteria outputs for scoring.

Generally, there are ten sets of criteria inputs/outputs, therefore, there will be ten pieces of outputs generated after the execution. In the scoring progress, each of the generated output will be compared with the corresponding criteria one to check whether they match with each other or not. If an output matches with its criteria one, the corresponding test case is scored as 10 points, otherwise, zero.

3. ENHANCED NEOESPA

3.1 System Procedure

We deploy SonarQube in the same server with neoESPA and developed several functions as bridges to connect the two systems as one. The student can submit his/her source code to the system for checking both the correctness and the code quality.

Figure 1 shows the procedure that the student submits his/her source code for the assignment to the enhanced neoESPA. At the front-end, the student can use the web browser to upload the source code to the compiler at the back-end. Once the compilation process is finished, an executable program is generated and sent for both code quality detection and correctness assessment.

The correctness assessment is similar to the older version of neoESPA we introduced in Section 2.2. We take advantage of existing functions to grade the generated outputs with predefined criteria ones. All of the scores of output matching will be added together to calculate the correctness score.

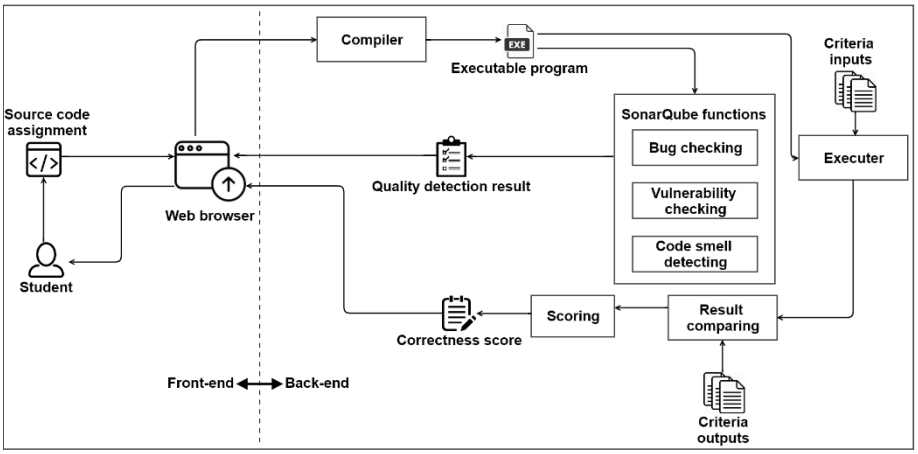

Figure 1. The code submission procedure in the enhanced neoESPA

In the code quality detection progress, the program is sent to the SonarQube functions for bug checking, vulnerability checking, and code smell detecting. The program will be tested under various built-in and predefined rules to detect it whether passes the rules or not. Since it is unnecessary to detect the code quality on the incorrect code, we design the procedure as detecting the code quality after the compilation is filtering those codes which contain error(s).

After both the correctness assessment and the code quality detection, the correctness score and the quality detection results are sent to the student through the web browser. The student can check his/her correctness score and the detected code quality result to find out which part of the code should be improved. We also imply the web interface of SonarQube for the instructor to locate the detected quality issues in the code.

3.2 Code Quality Detection

Figure 2 illustrates the progress of the code quality detection used in the proposed system. The sonar scanner takes the target program and analyzes it based on the detecting rules and plug-ins provided by the sonar server. The detected result can be generated for displaying in the web interface and storing it in the database.

The scanner scans the program code statically to detect whether there are violations related to the predefined programming rules. The rules are summarized based on the widely used coding guidelines, the common programming regulations, and the de facto programming patterns. The plug-ins such as FindBugs and CheckStyle strengthen the capability of code quality detection by extending more rules. The detected result can be accessed either by the web interface or the web API.

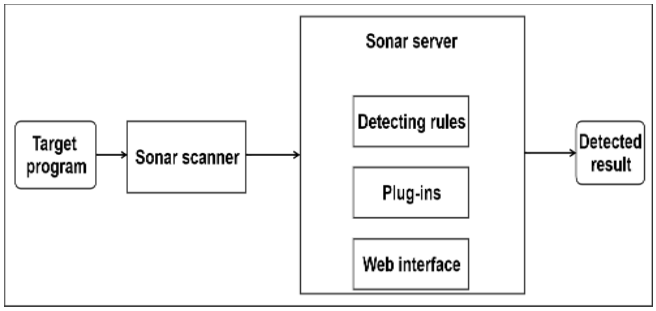

Figure 2. Progress of code quality detections

3.3 Data Collection

We collect two thousand codes from past programming courses to detect the code quality by using our proposed system to verify its practicality. We chose the Python codes as particular test targets since Python is the first programming language that undergraduate students learn in our department. It is important that finding and fixing the code quality issues at the beginning of learning programming since the issues may occur in other programming languages if they haven't been fixed in the first place.

Table 1. The assignments in Python programming course

Assignment #	Assignment name	Assessment concept
1	Difference of squares	The arithmetic operation, standard I/O
2	Weekly table	Basic grammar, functions
3	The most frequent character	Built-in methods
4	Max weight	Type conversion
5	Name value	Map
6	Stairways to heaven	Recursion
7	Uniqueness	Data operation
8	Word score	List comprehension

According to the features and the curriculum arrangement of Python, the course of Python programming is taught to the freshmen in their first semester. The assignments for this course are presented in Table 1. The assignments require specific concepts of Python and related programming skills. For each assignment, the student needs to develop his/her own program that takes an input and returns a corresponding output.

From the first to the third assignments in Table 1, the student needs to use basic instructions to develop simple interpreted programs to reach the requests. From the forth to the eighth assignments, the features such as data processing and use of the standard libraries are requested when the student develops the program.

Since it is an introductory course, there is no difficult concept such as abstraction and inheritance in the education of Python programming. However, the code quality issues in such simple programs are still necessary to be analyzed in case of the issues may occur in the more complex programs in the future.

4. EXPERIMENT

We calculate the issues that detected most frequently in the codes for the assignments we collect in Section 3.2. The experiment result is listed in Table 2. Each of the issues is described with the issue name, occurred times (detections), severity, and tagged type(s), respectively. We explain the reason for causing each issue and provide suggestions to solve it.

The first issue in Table 2 caused by assigning a local variable with a value but never used. This operation may waste computation resources. Also, it can lead to an error when overwriting it somewhere else. Removing the unused code can reduce the burdensome structure of the program and improve its efficiency.

The second issue caused by a complex body of a function. The complex control flow in a function can affect the performance of the program. It also makes the program difficult to maintain. Removing some of the control flow or split it into a new function may be an appropriate solution.

The third issue caused by setting the same consequents in all conditional expressions. Such an issue can reduce readability and maintenance. It is necessary to whether to add different operations or remove the same ones.

The fourth issue caused by using two continuous `if` expressions with the same consequents. Similar to the third issue, this mistake also leads to inefficient execution. Merging the two `if` statements is an effective solution to solve it.

The fifth issue caused by leaving an empty block in a conditional expression. The empty block may occur error since the unintended operations may drop into it. Removing the empty block can solve this issue.

The sixth issue caused by assigning a variable by itself. The operation such as `x = x` is meaningless and unnecessary since it doesn't change anything. Such a confused assignment should be removed to improve the readability.

The seventh issue caused by inserting extra operation after the `return` statement. Since the return statement jumps to another code block, the following operations won't be executed and become dead code. It is necessary whether to move the `return` to the position where after its following statements or remove the following ones.

The eighth issue caused by using a meaningless `pass` statement after a consequent. Since the if expression will be jumped out when the consequent was executed, it is unnecessary to call `pass`. Removing the return statement can solve this issue.

The ninth issue caused by calling the same variables on both sides of a binary operator. The conditions such as `y == y` will always return true and the consequent will always be executed. Whether removing the conditional expression or change the condition can solve this issue.

The tenth issue caused by using the duplicated condition in an `if-else if` expression. Since the expression is executed from the `if` statement, the consequent in the `else if` statement will never be executed and become dead code. Whether changing the condition in the `else if` statement or removing it can solve this issue.

Table 2. The most frequent issues in codes for the Python programming assignments

Issue #	Issue	Occurred times	Severity	Tag
1	Unused local variables should be removed	297	Code smell	Unused
2	The cognitive Complexity of functions should not be too high	130	Code smell	Brain-overload
3	Two branches in a conditional structure should not have exactly the same implementation	110	Code smell	Design, Suspicious
4	Collapsible `if` statements should be merged	62	Code smell	Clumsy
5	Nested blocks of code should not be left empty	21	Code smell	Suspicious
6	Variables should not be self-assigned	21	Bug	CERT
7	Jump statements should not be followed by dead code	17	Bug	CERT, CWE, MISRA, Unused
8	The `pass` should not be used needlessly	8	Code smell	Unused
9	Identical expressions should not be used on both sides of a binary operator	7	Bug	CERT
10	Related `if/else if` statements should not have the same condition	5	Bug	CERT, Pitfall, Unused

5. DISCUSSION

We realize that most of the issues found in the experiment are novice-like. As aforementioned, the programming language that the freshmen firstly learn is Python, it is easy to cause lots of incorrect or inefficient operations when learning a new

programming language since the students are not familiar with it. However, such novice mistakes should be fixed immediately since the mistakes may mislead the students to keep a bad programming style when they learn other more complex languages.

We recognize that the students pay lots of attention to solving the assignment problems to generate correct answers but omit the code quality issues when they conducting program codes. The instructor should emphasize the concepts of code quality in programming education since such a skill is helpful for the students when they graduate from university to seek jobs.

The proposed system currently only supports the code quality detection for Python. The code quality of programming assignments written in other widely used programming languages such as C/C++ and Java are also necessary to be detected. The analysis of the code quality among various programming languages can discover constructive contribution for improving the programming education.

6. CONCLUSION

This paper presents an enhanced online programming judge (OPJ) that can not only assess the correctness of the programming assignment code but also detect its quality. The proposed system implies a code quality management tool named SonarQube on our OPJ called neoESPA. In the experiment, we execute two thousand Python assignment codes in the enhanced neoESPA to detect the most frequent code quality issues. The result shows that most of the issues are novice-like and should be fixed immediately to prevent such mistakes from happening again when the students learning other programming languages.

In the future, we are going to keep developing the proposed system to archive the code quality detection for more programming languages. We are also going to analyze the differences and distributions of code quality issues among various languages to help the instructor to improve programming education.

7. ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2014-3-00035, Research on High Performance and Scalable Manycore Operating System)

8. REFERENCES

[1] Kalogeropoulos, N., Tzigounakis, I., Pavlatou, E. A., & Boudouvis, A. G. 2013. Computer-based assessment of student performance in programming courses. *Computer Applications in Engineering Education*, 21, 4 (Oct. 2013), 671-683.

[2] Gulwani, S., Radiček, I., & Zuleger, F. 2018. Automated clustering and program repair for introductory programming assignments. In *ACM SIGPLAN Notices*. 53, 4, 465-480. ACM.

[3] Chilana, P. K., Alcock, C., Dembla, S., Ho, A., Hurst, A., Armstrong, B., & Guo, P. J. 2015. Perceptions of non-CS majors in intro programming: The rise of the conversational programmer. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE.

[4] Keuning, H., Jeuring, J., & Heeren, B. Towards a systematic review of automated feedback generation for programming exercises. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 41-46). (2016, July). ACM. DOI=10.1145/2899415.2899422

[5] Cedazo R, Garcia Cena CE, Al‐Hadithi BM. A friendly online C compiler to improve programming skills based on student self‐assessment. Computer *Applications in Engineering Education*. 2015;23(6):887-896. doi: 10.1002/cae.21660

[6] Shamsi FA, Elnagar A. An intelligent assessment tool for students' java submissions in introductory programming courses. *Journal of Intelligent learning systems and applications*. 2012;4(01):59. doi: 10.4236/jilsa.2012.41006

[7] Wasik S, Antczak M, Badura J, Laskowski A, Sternal T. Optil. io: Cloud based platform for solving optimization problems using crowdsourcing approach. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion* (2016), ACM, pp. 433-436. doi:10.1145/2818052.2869098

[8] Restrepo‐Calle F, Ramírez Echeverry JJ, González FA. Continuous assessment in a computer programming course supported by a software tool. *Computer Applications in Engineering Education*. 2019;27:80-89. doi:10.1002/cae.22058.

[9] Silva D, Nunes I, Terra R. Investigating code quality tools in the context of software engineering education. *Computer Applications in Engineering Education*. 2017;25(2):230-241. doi:10.1002/cae.21793

[10] Kan SH. Metrics and models in software quality engineering. *Addison-Wesley Longman Publishing Co., Inc*. 2002;23-24

[11] Lewis, W. E. (2017). *Software testing and continuous quality improvement*. Auerbach publications.

[12] Liu, X., Kim, Y., Cho, H., & Woo, G. (2015). NeoESPA-A New Evaluation System for Programming Assignments. In *Proceedings of the 14th International Conference on Electronics, Information, and Communication (ICEIC 2015)*.

[13] Martignano, M., Jung, A., & Ihmann, T. (2015). Source code analysis of flight software using a SonarQube based code quality platform. *Ada User Journal*, *36*(2), 99.

[14] Tollin, I., Fontana, F. A., Zanoni, M., & Roveda, R. (2017, June). Change prediction through coding rules violations. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (pp. 61-64). ACM.

[15] Built-in Rule Tags, https://docs.sonarqube.org/latest/user-guide/built-in-rule-tags/, Accessed 10 October 2019.

[16] Stegeman, M., Barendsen, E., Smetsers, S. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research* (2016) ACM. pp. 160-164.

[17] Altadmri A, Brown NC. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (2015) ACM, pp. 522-527