Analyzing the Code Quality Issues in Java Programming Assignments

Xiao Liu[1], Yeoneo Kim[1], Junseok Cheon[1], Taekwang Hur[1], Sugwoo Byun[2], Gyun Woo[1]
Dept. of Electrical and Computer Engineering, Pusan National University[1],
Dept. of Computer Science, Kyungsung University[2]
e-mail: {*liuxiao,yeoneo,jscheon,vhxpffltm,woogyun*}*@pusan.ac.kr*[1], *swbyun@ks.ac.kr*[2]

Abstract

This article detects and analyzes the code quality issues (CQIs) in students' Java programs to discover what kinds of CQIs affect the students' programming quality most frequently and provides possible solutions to fix such issues. We use a code quality detector named SonarQube to detect two thousand programs submitted by students in the past Java programming course. We generate the solutions based on the feedback of SonarQube and re-detect the programs after modifying them to verify the improvement of code quality. The result indicates the CQIs in students' programs are diverse and numerous. The solutions for fixing such issues are efficient. The result drives us to look forward to apply the code quality detection as a sub-function of an online judge to help students to improve their programming quality by themselves in the future.

I. Introduction

The correctness of the programs' outputs have been considered as the main aspect when assessing the programming assignments in programming education. The lack of code-quality-related education in the current programming courses causes students don't pay enough attentions on their programming quality [2]. However, since the industry values and requires to guarantee high software quality more than ever [3], it is important to help students to discover and correct the code quality issues (CQIs) of their programs when they are learning a new programming language.

The researchers in programming education have noticed the code quality concept has not been taught in many programming courses [4]. In order to collect criteria of measuring CQIs in students' programs, a study suggests interviewing instructors, students, and professionals to obtain measurements for assessing code quality [9]. Such approach can't assure the criteria covers all CQIs since the interview may be affected by the subjective wills of interviewees. In order to detect CQIs in students' programs, there are studies adopting human graders to assess students' code [5, 6]. Such approaches require lots of human work and the consistency of the graders can affect the accuracy of the code quality evaluation.

In this paper, we use SonarQube to automatically detect the CQIs in students' programs. The thousands of coding rules defined in SonarQube can help us to identify all CQIs with suggestions of improvement. We inspected two thousand programs submitted by students in the past Java courses. We conduct a quantitative analysis through the detected CQIs to discover the most frequent ones in students′ code. We also provide solutions for fixing such CQIs. In order to verify the efficiency of our study, after modifying a certain number of the programs based on the solutions, we re-inspected the programs to check the improvements of the programs' code quality.

The rest of this paper is organized as follows. Section 2 describes the background such as the concepts of code quality and SonarQube. Section 3 presents the methodology of our study including system development, data collection, and experiment design. Section 4 demonstrates the experiment result. Section 5 discusses the challenges and insufficient of the study. We conclude and address future work in Section 6.

II. Background

2.1 Code quality

The aspects of the term of code quality such as capability, usability, performance, reliability, and maintainability affect the features of program such as efficiency, vulnerability, and security [7]. A program with a good quality can extend its lifetime and reduce extra costs of maintenance. In contrast, a program with a bad quality may lead to critical problems in execution such as wasting of computational resources, data breach, and unexpected shutdown. The most appropriate way to develop a high code quality program is writing code with correct coding style [1]. Therefore, to help the students improve their abilities of writing qualified code is an indispensable work in programming education.

2.2 SonarQube

As a widely used platform for code quality management, SonarQube consists of the server and the scanner [8]. The server provides an environment to execute functions such as code management, detection result management, and API handling. The scanner detects the quality of the uploaded code and generates a detected result. SonarQube supports features such as bug checking, vulnerability checking, and code smell detecting, which can help us to detect the CQIs in students' programs. Besides, the rules used in SonarQube for quality detection can be extended, which allows us to add particular rules to build more suitable CQIs.

The basic mechanism of quality detection in SonarQube is analyzing the target code statically with several quality rules to detect whether the code passes the rules or not, and generating the detected result. The rules used in SonarQube contains the programming guidelines of each corresponding language, the common programming regulations, the de facto coding patterns, and the wildly used coding conventions. These rules also have been used in industries to standardize the quality of software. Therefore, it is beneficial for students when they getting jobs after graduation if they keep practicing programming skills under the quality rules.

Ⅲ. Methodology

3.1 System procedure

Figure 1 illustrates the procedure of detecting students' CQIs. In step 1, the user can upload the students' source code files through web browser to the system server. Step 2 represents the scanner detects the CQIs under the SonarQube functions such as bug checking, vulnerability checking, and code smell detecting. After the detection, the CQIs result is generated and sent back to user through web browser in step 3.

Figure 2 presents an example of detected CQIs result of a student's program. The CQIs are marked by red wavy lines and the corresponding description of coding rule violation is placed under each line. Each CQI can be solved through modifying the code by referring the description.

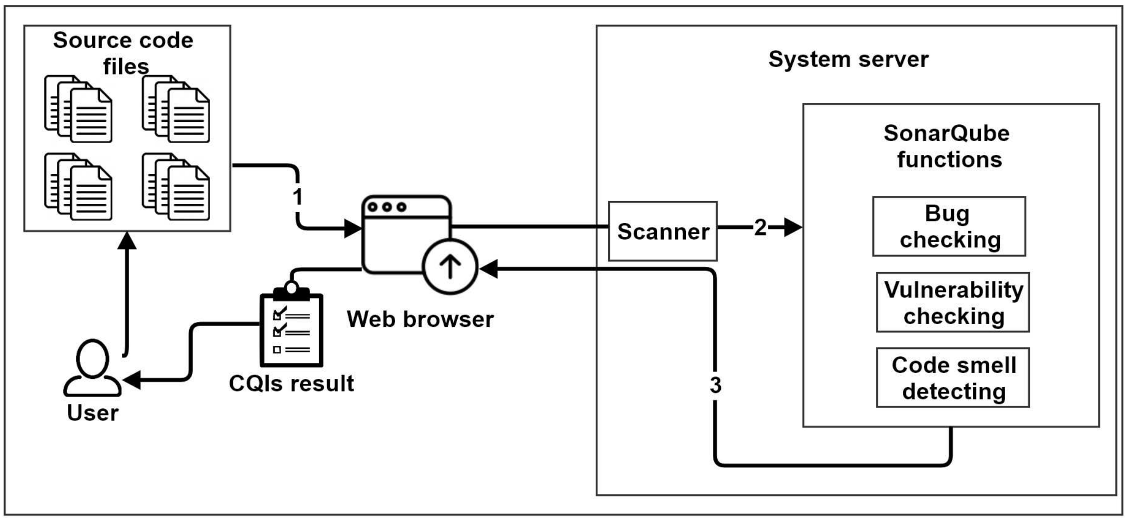
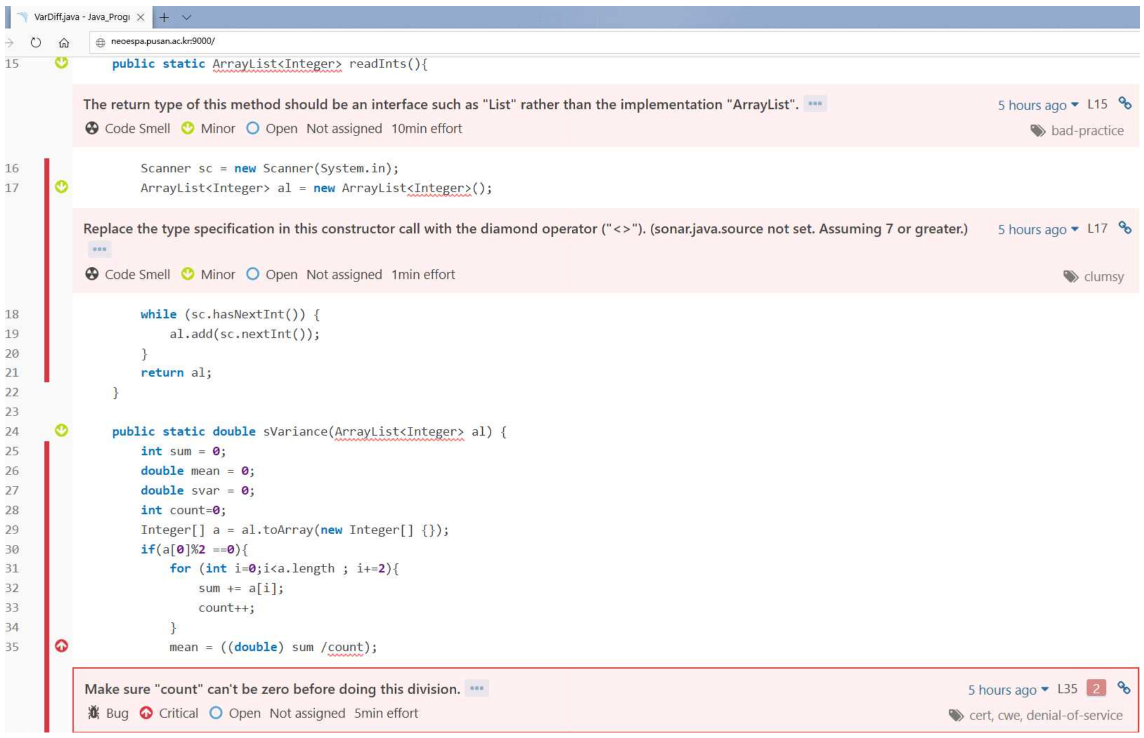


Figure 1 The procedure of detecting CQIs in students′ programs



```
    public static ArrayList<Integer> readInts(){
```

The return type of this method should be an interface such as "List" rather than the implementation "ArrayList". ··· 5 hours ago L15
Code Smell Minor Open Not assigned 10min effort bad-practice

```
        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> al = new ArrayList<Integer>();
```

Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) ··· 5 hours ago L17
Code Smell Minor Open Not assigned 1min effort clumsy

```
        while (sc.hasNextInt()) {
            al.add(sc.nextInt());
        }
        return al;
    }

    public static double sVariance(ArrayList<Integer> al) {
        int sum = 0;
        double mean = 0;
        double svar = 0;
        int count=0;
        Integer[] a = al.toArray(new Integer[] {});
        if(a[0]%2 ==0){
            for (int i=0;i<a.length ; i+=2){
                sum += a[i];
                count++;
            }
            mean = ((double) sum /count);
```

Make sure "count" can't be zero before doing this division. ··· 5 hours ago L35 2
Bug Critical Open Not assigned 5min effort cert, cwe, denial-of-service

Figure 2 An example of detected CQIs in a student′s program

3.2 Data collection

We choose two thousand programs submitted by the students in past Java programming course. The programs were developed for assignment submissions. We applied several assignments during the course covering the concepts such as fundamental syntax, use of built-in methods, types and classes, exception handling, and concurrent computing. Since the programs cover the entire learning stages, we can find out all CQIs occurred in different levels.

3.3 Experiment design

We aim to detect the most frequent CQIs occurred in students' programs. For each kind of CQI, we provide an appropriate solution to solve it. In order to verify the efficiency of the solution, we resubmit the modified program to our system and check the improvement of code quality.

Ⅳ. Result

The most frequent CQIs detected in students' programs are listed in Table 1. We give a short description of the CQI, the occurred reason, the frequency (Freq.), its severity defined by SonarQube, and the corresponding solution, respectively. Most of the CQIs are caused by misuse of variables and conditional expressions. Such CQIs indicate students only paid attention on the correctness of the program's output but ignored to write code properly.

We modified a certain number of the programs by referring the solutions and resubmit them to our system. The result shows all the CQIs can be solved by the solutions.

Table 1. Most frequent CQIs in students' programs

No.	CQI	Occurred reason	Freq	Severity	Solution
1	Unused local variables should be removed	Declaring a local variable that never been called	297	Code smell	Deleting the unused code
2	The cognitive Complexity of functions should not be too high	The body of a function is too complex	130	Code smell	Splitting the complex function into a new one
3	Two branches in a conditional structure should not have exactly the same implementation	Setting the same consequents in all conditional expressions	110	Code smell	Either removing the same condition or changing it
4	Collapsible *if* statements should be merged	Using two continuous *if* expressions with the same consequents	62	Code smell	Collapsing the two expressions into one condition
5	Nested blocks of code should not be left empty	Leaving an empty block in a conditional expression	21	Code smell	Either removing the corresponding expression or adding new statements in it
6	Variables should not be self-assigned	Assigning a variable by calling itself, such as $x = x$	21	Bug	Removing the corresponding variable assignment
7	*jump* statements should not be followed by dead code	Adding additional statements after the return statement	17	Bug	Removing the code block after the *jump* statement
8	*pass* should not be used needlessly	Using a meaningless pass statement after a consequent	8	Code smell	Removing the unnecessary *pass* statement
9	Identical expressions should not be used on both sides of a binary operator	Calling the same variables on both sides of a binary operator such as $x == x$	7	Bug	Either removing the expression or changing one of the variables to another value
10	Related *if-else if* statements should not have the same condition	using the duplicated condition in an *if-else if* expression	5	Bug	Either removing one of the same conditions or changing it to a different one

Ⅴ. Discussion

We recognize the lack of the code-quality-related education causes numerous CQIs in students' programs. Our solutions can effectively solve the CQIs to improve the quality of the programs. However, the experiment is conducted without the participations of students which can not assure the same efficiency can be performed in the real programming course. Additionally, the use of the proposed system requires knowledge of using SonarQube, which may stress students learning such knowledge if the system is applied in a real course. Embedding the system into an online programming judging system to let students check the CQIs in their programs may help them to improve programming quality in a more efficient way.

Since students are learning programming skills through different programming courses, it is necessary to analyze the relations of CQIs among different programming languages. For example, if Python and C were taught before Java, it can be helpful in discovering the same CQIs occurred in the programs written in the former learned languages. Then we can research on improving students' programming quality earlier to help them avoid the

CQIs in the future learning programming language.

VI. Conclusion and future work

We conduct an feasibility research on detecting code quality issues (CQIs) in students′ Java assignment programs. The result indicates the lack of code-quality-related education causes students to make lots of mistakes in writing codes. The detected CQIs can be efficiently solved by referring the solutions we suggested.

In order to help students to improve programming quality in a more convenient way in the future, we will work on embedding the proposed system into an online programming judging system during a practical programming courses to let students detect and solve the CQIs by themselves.

Acknowledgement

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).

Corresponding author: Gyun Woo (woogyun@pusan.ac.kr)

References

[1] Stamelos I, Angelis L, Oikonomou A, et al. "Code quality analysis in open source software development", Information systems journal, 12(1): 43-60, 2002.

[2] Zaw K K, Hnin H W, Kyaw K Y, et al. "Software Quality Metrics Calculations for Java Programming Learning Assistant System", IEEE Conference on Computer Applications, 1-6, 2020.

[3] Silva D, Nunes I, Terra R. "Investigating code quality tools in the context of software engineering education", Computer Applications in Engineering Education, 25(2):230-241, 2017.

[4] Kirk D, Crow T, Luxton-Reilly A, et al. "On Assuring Learning About Code Quality", Twenty-Second Australasian Computing Education Conference, 86-94, 2020.

[5] Stegeman M, Barendsen E, and Smetsers S. "Towards an empirically validated model for assessment of code quality", 14th Koli Calling international conference on computing education research, 99-108, 2014.

[6] Costantini U, Lonati V, Morpurgo A. "How plans occur in novices' programs: A method to evaluate program-writing skills", 51st ACM Technical Symposium on Computer Science Education, 852-858, 2020.

[7] Kan S H. "Metrics and models in software quality engineering", Addison-Wesley Longman Publishing Co., Inc.; 2002;23-24

[8] Kozik R, Choraś M, Puchalski D, et al. "Platform for software quality and dependability data analysis", International Conference on Dependability and Complex Systems, 306-315, 2018.

[9] Börstler J, Störrle H, Toll D, et al. "I know it when I see it - Perceptions of Code Quality". ITiCSE Conference Working Group Report, 70-85, 2018.