Hindawi Publishing Corporation
ISRN Software Engineering
Volume 2013, Article ID 805963, 8 pages
http://dx.doi.org/10.1155/2013/805963

Research Article

Feedback for Programming Assignments Using Software-Metrics and Reference Code

Pardha Koyya, Young Lee, and Jeong Yang

Department of Electrical Engineering & Computer Science, Texas A&M University-Kingsville, TX 78363, USA

Correspondence should be addressed to Young Lee; young.lee@tamuk.edu

Received 30 September 2013; Accepted 22 October 2013

Academic Editors: C. Calero and Z. Shen

Feedback for student programming assignments on quality is a tedious and laborious task for the instructor. In this paper, we make use of few object-oriented software metrics along with a reference code that is provided by the instructor to analyze student programs and provide feedback. The empirical study finds those software metrics that can be used on the considered programming assignments and the way reference code helps the instructor to assess them. This approach helps the instructor to easily find out quality issues in student programs. Feedback to such assignments can be provided using the guidelines which we will be discussing. We also perform an experimental study on programming assignments of sophomore students who were enrolled in an object-oriented programming course to validate our approach.

1. Introduction

Assessment of students programming assignments are mostly done by using certain criteria like functionality, design, and programming style. Moreover, giving feedback on students' assignments is a hectic task because the instructor needs to inspect all student assignments. Many computer-aided approaches (CAAs) have been proposed for assessing student assignments and giving feedback [1]. Using software metrics to assess student programs is one of those several approaches used in CAAs.

Software metrics are measures of certain aspects of a program that help to analyze and control it is quality. Researchers have found a large number of software metrics that can analyze different aspects of the source code. These software metrics can also be used by the instructor to analyze the students programing assignments and give feedback. Most of these software metrics used were related to complexity and size of the programs [2–5]. Cohesion and coupling are certain aspects of a program that measure it is relatedness and dependency. In this research, we will find out whether some metrics relating to cohesion and coupling are capable of assessing student programs and also reveal the results of an experiment that was conducted on the student assignments.

As we proceed, we will be answering three questions pertaining to our research.

(1) Can the object-oriented metrics assess student programs?

(2) How can these metrics with reference code help the instructor in analyzing student programs?

(3) Will this approach help the students?

We have organized this paper into different sections. In Section 2, we perform a review about the previous related research conducted in this area. The next Section 3 explains our present idea on using software metrics and reference code to provide feedback on student programming assignments. The first question will be answered in Section 4.1 by obtaining and analyzing the metric values from the CKJM tool. In Section 4.2, we perform an experimental study on sophomore student assignments from an object-oriented software engineering course. Section 4.3 provides answers to questions (2) and (3) using surveys from the grader and students finding out whether our assumed software metrics and reference code are able to assess student programs.

2. Literature Review

The main approaches towards assessing student programming assignments can be categorized into formative and summative. Formative assessments provide feedback to the student while learning, whereas summative assessments evaluate the learning process after completion. Another approach is the diagnostic assessment which is used to identify student learning difficulties and areas of strength [2].

An automated programming assessment system (APAS) is capable of analyzing and assessing student programs thereby reducing the instructor's workload. Presently, there are several APASs used in many computer courses. Along with program analysis some of these APASs provide feedback and some do not. Feedback is required by the students on their programs for their improvement. In this study, we focus on some approaches that used static assessments centering the idea of using software metrics, model codes and providing feedback to the students.

Software metrics are considered to be general measurements which characterize the computer programs [1]. Mengel and Yerramilli [6] considered software metrics to be indicators of student performance and also instructional development. The software metrics are capable of measuring both the static and dynamic features of a programming assignment. The dynamic features include correct functionality, efficiency, student testing skills, and searching for special features which answer specific problems related to dynamic execution of programming assignments. Features like coding style, software metrics, programming errors, design, and special features related to program structure come under static assessments [1]. It also includes diagram analysis, keyword detection, and plagiarism detection.

The static approach does not require that the code be compiled and executed whereas dynamic approach requires so. Assessment is directly done on the code in static approach. The readability and maintainability factors of a program are assessed by the programming style feature. They are highly enhanced by the criteria like comments, indentation, line spacing, program layout, and meaningful variable names. Checkstyle is one of the tools that checks the Java code for these criteria. Error detection in a program is done using the syntax and semantics of the programming language. A programing language's syntax is written against the grammar specification of that language. The semantics of a language forms the logical correct statements that conform to the grammar of the language. These errors are generally detected by the IDE (interactive development environment) of the language. Eclipse is one of such IDEs for Java. Another method of static assessment is the structure analysis which is done by matching the structures of programs. Generally the assignment's structure is compared with the model programs structure. Program analysis, transformation, and program matching are used in this approach [7]. Size, structure, and statements are matched with the model programs for grading. The closer the structure to the model program's structure, the higher the grade the student gets. With the increase in the size and complexity of the program the required number of model programs also increases in this approach. This is one of the fully automatic approaches for assessing student assignments. There are semiautomatic approaches which include continuous intervention of the instructor with the system for assessing the student assignments. The instructor may be asked to write the test cases against which all the student assignments will be checked. Online Judge [8] is one of such systems that is used in program assessments for correctness and efficiency. It requires several test cases which are generally provided by the instructor. Then, it compiles and runs the assignments under certain limits of time and memory. Robustness of programs can also be determined using this system by giving out extreme test cases. The outputs of the programs are compared to prespecified answers using simple string matching. Maintainability of the code cannot be determined by this system and is left to the tutor. The grade and feedback given by the tutor will be shown to the student.

The process of giving feedback was also automated in the systems which evolved later. FrontDesk [9] is on such systems providing a web interface to students for assignment submission and had objective and subjective testing systems. The objective testing system has test suites which communicate with a testing center. The testing centers are installed on multiple workstations and reduce the workload of conducting the objective tests. For developing the test suites FrontDesk provides a JUnit [10] extension which is widely used for Java programs. These test suites output the result in XML format which conforms to the XSD schema defined by FrontDesk. The subjective testing system provides feedback to the students using the help of the instructor. It provides categories in a tree view which are required to be filled by the instructor. The instructor needs to determine the points that the student acquires for each category. The grading schema contains predetermined remarks which have a point value, comments and a message to the student.

Software metrics have not been widely used in educational institutions for assessing student programs [1]. These metrics provide numerical values that measure certain characteristics of the code and form the basis for determining student programs through evaluation and comparing. Moreover, these metrics can be obtained automatically using several tools. Bowman and Newman [11] had achieved positive results in using the software metrics as programming training tool for students. They have used different dimensions of the complexity metric to train the students in programming. Means and deviations of each complexity metric and development times were used to analyze the results. McCabe's complexity metric was proposed in 1976 which measured the cyclomatic complexity of a program [12]. The cyclomatic complexity is the measure of the different control paths in a program. Assyst [4] was one such APASs that used the complexity metric to assess student assignments. The complexity metric is calculated for both the student assignment and the model code and the difference is used to grade the assignments. It also considers the correctness, efficiency, style, and data coverage of the programs. The data coverage was measured using the test effective ratio (TER1) which is the number of statements executed at least once divided by

the total number of executable statements in the program. Leach [13] used the Halstead effort metrics which are based on counting different attributes such as number of operands and operators in a program to assess the student programs. Halstead measure and McCabe's measure reference different dimensions of complexity and showed success in assessing student programs [1, 4]. Another important metric which can be used to assess student programs is the LOC (lines of code) which is the total number of executable statements in a program. Hung et al. [3] proved that a strong correlation existed between LOC and the debugging effort in student programs. The program analyzer [3] considered the total statements, lines of code, and the comment lines to assess the student's program. The compilation errors were divided into 11 types and the students were given feedback using the frequency distribution of different types of compilation errors. The feedback was only limited to the compilation problems that occurred in the student programs.

Cardell-Oliver [2] used software metrics for both diagnostic assessment and formative assessment of novice programmers. Program size was considered one of the factors for diagnostic assessment which required a software metrics for analysis. In addition to the LOC metric it used other metrics like noncomment lines of code, number of methods, number of fields and their subsets for program assessment. The idea of using a model program was also implemented in this research. Students were classified into different categories by calculating the deviation of their program metrics from the model code's metrics. Students were also allowed to use tools like JUnit, checkstyle, and PMD which give feedback on correctness and code quality. These tools provided formative feedback to the students on their programs. The result of the experiment was that software metrics helped novice programmers to improve their learning.

Fuzzy logic was used in assessing student programs by applying it on software metrics and test cases [5]. This semi-automatic approach also had an ideal solution with which the student programs were compared. Metrics related to the number of method calls, recursive calls, arrays, variables, blocks, value parameters, and reference parameters were calculated for comparison. These software metrics formed the basis for obtaining the fuzzy representation of both the ideal algorithms. The ideal solution along with the lowest and highest possible values for each metric was provided by the instructor which were used for fuzzy representation. Each of the obtained metric value was normalized before comparison. The structural assessment was done by measuring the similarity degree between the fuzzy representation of the ideal solution and the student programs. Feedback to the students was provided using a fuzzy logic rule-based system.

As mentioned above we focused our study on some approaches that used software metrics, model codes, and feedback to students. Among these approaches, less focus was given to object-oriented metrics in assessing student programs. So, in this research we will try to find out whether object-oriented metrics can help in assessing student programs.

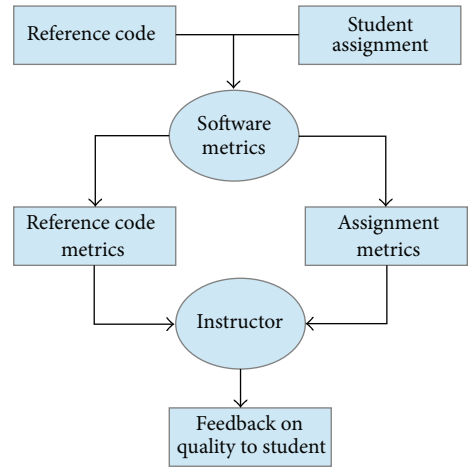

Figure 1: Proposed approach.

3. Approach

Chidamber and Kemerer [14] proposed six object-oriented metrics that determine the structure of a program. Among these, we consider the metrics which measure the coupling and cohesion properties of a program. Coupling is a property of a class that determines it is interdependencies with other classes. Cohesion is the measure of relatedness of functions in a class and complexity metrics determine the complexity of a program. We also consider the McCabe cyclomatic complexity metric that has been successfully used on student assignments. Figure 1 depicts our approach.

Our idea is to accompany the software metrics with a reference code which is provided by the instructor and analyze the assignments for providing feedback on their quality. We assume that all assignments are functionally sound and tested before their metrics are calculated. We use the CKJM (Chidamber and Kemerer Java Metrics) [15] extended tool to compute the metrics both from the assignments and reference code. This tool computes the object-oriented metrics along with few other metrics of coupling and cohesion that will be used in our research. The instructor can find the assignments that are comparatively lower in quality by the class-class metrics comparison and inspect them for providing a feedback. The instructor will be provided with the guidelines that mention the reasons for which the metrics are varied. Instead of going through every assignment for quality analysis, the instructor can only inspect the assignments which have higher deviations from the reference code. Feedback relating to the metrics that have higher deviations from the reference code is provided to the student using the guidelines provided to the instructor about the metrics. As the student does not have knowledge about the software metrics, there exists a huge gap between the metric results and the student's understanding on the feedback from the metrics analysis. Here, the instructor bridges this gap by providing the feedback using the guidelines in such a way the student understands.

Finally, we compare the time needed to analyze the assignments by the instructor with and without our approach

for the issues relating to cohesion and coupling complexity. We also take a survey from the students on how well the feedback can be comprehended by providing them with the reference code.

4. Empirical Study

We perform the empirical study to find two critical issues that relate to the research. The first issue corresponds to whether the quality metrics of coupling and cohesion are capable of assessing student programs. Most of the metrics are insensitive to student assignments because of their smaller size. Generally, students are given assignments that can be accomplished using few classes. Hence, most of the object-oriented metrics which were obtained from the CKJM tool did not have much variation. Some of the metrics values were found to be similar for all the assignments and could not be used to detect variations in the assignments from the reference code. For example, the inheritance and coupling metrics were similar in all the assignments as they were all targeting a common functionality and were too small in size.

The second issue focuses on the way that these metrics help the instructor to analyze the student assignments for giving the feedback. The metric values obtained from the tool are just a measure of their kind from the way the code is written. They are alone insignificant and resembling nothing. It is difficult to decide whether the metric value is good or bad from the established standards. For example, the LCOM3 (Lack of Cohesion among Methods) value which ranges from 0 to 2 is considered harmful if the value exceeds 1. These standards are too general for the smaller sized student assignments. Hence, we need to establish another standard for determining the metrics obtained from the student assignments. We procure this standard in the form of a reference code which is provided by the instructor. The reference code's metric values are used as a standard to compare the student assignments.

4.1. Methodology. We have considered a sample of actual student assignments which belonged to the sophomore students of Computer Science who were enrolled for the object-oriented programming course. These students had basic knowledge on object-oriented programming such as creating a class, creating objects for a class, and invoking members of another class. They were given a Banking Assignment in which they were suggested to create two classes: one as the Main class and the other as an Account class. Their program must be able to display a menu which consisted of the basic banking activities for deposit, withdraw, transactions, interest, showing balance, getting interest rate, and added interest. The user should be able to input the options from the menu to the program for the desired operation. The instructor was asked to provide a reference code for the given assignment which will be used in our approach. The experiment was performed on 11 student assignments which were sound in their functionalities. We will now try to optimize the student's programming approach towards the same functionality using our approach of software metrics and reference code.

The CKJM tool was used to obtain the metrics from the student assignments and the reference code. This tool along with the Chidamber and Kemerer metrics [14] calculates various other metrics. Among them, we consider the metrics related to cohesion and coupling and apply them on student assignments. The coupling metrics, namely, fan in or afferent coupling (Ca) and fan out or efferent coupling (Ce) of classes are the measure of number of references made to and from a class. The CBO (coupling between object classes) is the number of classes to which a class is coupled and the RFC metric is the measure for the methods that are invoked when an object of the class is created. The cohesion metrics CAM and LCOM3 (a variant of LCOM) measure the cohesiveness of a class. The CAM metric calculates cohesion by considering the types of parameters that are used by the methods. The LCOM3 in-turn calculates the cohesion of the class by considering the effective usage of the class attributes. The metrics for Ca, Ce, and CBO did not vary among the assignments. Also the IC (inheritance coupling) was similar for all the assignments. This is because all the assignments had similar number of classes and functionality. We have found the coupling complexity metric RFC (response for class), the cohesion metrics CAM (Cohesion Among Methods) and LCOM3 (lack of cohesion among methods) to be sensitive on the considered student assignments. Other such metrics were CC (cyclomatic complexity), LOC (lines of code), and WMC (weighted methods per class). For now, we will be focusing on the metrics other than LOC as it has been a successfully used metric on student assignments. The WMC metric which is the measure of all the methods in a class did not show much significant variations as all the assignments had to achieve the same functionality using the methods described in the assignment's problem statement. Therefore, the WMC and LOC metrics were not considered for our study. The CC metric which has been widely used on most APASs will be considered in our study.

We now compare the CAM, LCOM3, RFC, and CC metrics of the assignments and reference code for deviations. The grader was given guidelines on the four considered metrics on how they varied, the possible reason for their variation, and a suggested feedback to the student. The guidelines are useful to the grader to compare the assignments with the reference code.

Guideline 1. Higher values of RFC may be possible due to the presence of too many methods in the class.

Suggested Feedback 1. The student may be asked to modularize the code more effectively.

Guideline 2. Lower CAM values may be possible due to the use of methods/parameters which aren't required.

Suggested Feedback 2. The student may be suggested that the code be programmed using less methods than used.

Guideline 3. Higher LCOM3 values may occur due to the use of un-required/improper use of attributes.

```
System.out.println("Assignment 5\nGaylon Taylor\nK00257327");
do{

    System.out.println("\n              Menu                              ");
    System.out.println("-------------------------------------------");
    System.out.println("A) Display the account balance");
    System.out.println("B) Display the number of transactions");
    System.out.println("C) Display interest earned for this period");
    System.out.println("D) Make a deposit");
    System.out.println("E) Make a withdrawal");
    System.out.println("F) Add interest for this period");
    System.out.println("G) Exit the program");

    System.out.println("\nEnter your choice > ");
    action=keyboard.next().toUpperCase().charAt(0);
    switch(action)
```

Figure 2: Sample of a student programming assignment.

```
//statements to do the actions the user selected
if ("A".equals(bank_choice))
{
    balance=p1.show_balance();
    JOptionPane.showMessageDialog(null,"Your balance is $"+twoDigitsPastPoint.format(balance));
}
if ("B".equals(bank_choice))
{
     No_of_transactions=p1.gettransactions();
     JOptionPane.showMessageDialog(null,"You have made "+No_of_transactions+" transactions");
}
if ("C".equals(bank_choice))
{
    interest=p1.showinterest();
    JOptionPane.showMessageDialog(null,"Your interest balance is $"+twoDigitsPastPoint.format(interest));
}
if ("D".equals(bank_choice))
{
    String string_deposit = JOptionPane.showInputDialog("Please enter your deposit\n");
    double deposit=Double.parseDouble(string_deposit);
    p1.add_money(deposit);
    p1.addtransactions();
}
```

Figure 3: Sample of a student programming assignment.

Suggested Feedback 3. The student may be suggested that the code be programmed by more effective use of attributes.

Guideline 4. Higher CC values may be possible due to the presence of many paths in the code logic.

Suggested Feedback 4. The student may be suggested to carefully use the control statements.

4.2. Results. The values of the sensitive metrics on smaller assignments, namely, RFC, CAM, LCOM3, and CC, obtained from the CKJM tool were analyzed for each of the student assignments with the reference code. We have also suggested a feedback for the metrics which was evaluated by inspecting the parts of the assignment causing the deviation of metrics.

4.2.1. RFC. The RFC metric is the measure of number of different methods that can be executed when the object of the class receives a message. Higher RFC of a class indicates that the overall class design is complex. Therefore, to improve the class design one must reduce the number of methods that are invoked by a class object on receiving a message.

The assignments in this study had two classes one as the Main class and the other as an Account class to perform the user's banking functionalities depending on the input of the user's choice. There was not significant difference among the classes from the reference code. The Main class of the reference code was used to invoke the methods in the Account class using an object. The Account class consisted of the methods to display menu, deposit money, withdraw money, obtain transactions, calculate the interest, display balance, and obtain the interest value.

Among the student assignments, some showed significantly higher RFC in the Main class than the Account class. On inspecting all such assignments we found that the logic for displaying the menu and taking input from the user was found in the Main class. Figure 2 is one of the assignments in which the student displayed the Banking menu from the Main class. This approach to the solution increased the RFC count among the student assignments which leads to the God class problem. From the collected sample, we found 72% of the assignments to be having significantly higher RFC values for the Main class. Therefore, in such cases the instructor can inspect such assignments and can feed back the students to improve the modularity of the classes in the solution code using the suggested feedback 1.

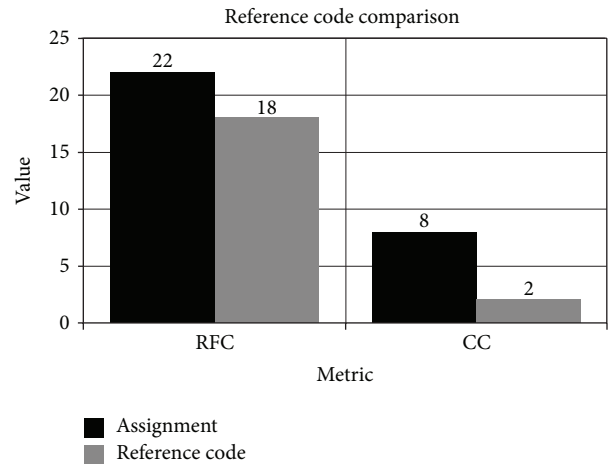

Figure 4: Comparison graph of assignments with reference code.

4.2.2. Cyclomatic Complexity. Cyclomatic complexity is the measure of linearly independent paths in the program. The more the CC value the more the complexity of the program is. In this study, the Account class of both the reference code and the assignments did not show much variation. So, we have considered the Main class's CC for analyzing the quality of the assignments.

As the Main class had to invoke the Account classes' methods depending on the user's choice, a decision statement was to be used. The reference code's Main class used the switch statement for the required functionality. On examining the CC metric values of student assignments, we found 36% of them to be having significantly higher complexities than that of the Main class. All of these assignments used a different decision statement for the same functionality which increased their complexities. They used an if-else-if ladder instead of the switch statement which would be more effective for the assignment's solution. Figure 3 is one such assignment where an if-ladder has been used making the code complex. Therefore, the instructor, in this case, can provide feedback to use a different decision statement which would be more effective.

Figure 4 shows the comparison of the RFC and CC metrics of a student assignment and the reference code.

4.2.3. CAM. The CAM metric measures the cohesion among the classes methods by considering the types of parameters

```
18   //Create constructor
19   public Banking(){
20        balance = 0;
21        interest = 0;
22        transactions = 0;
23        interestRate = 0;
24   }
25
26   public Banking(double int_balance){
27        balance = int_balance;
28        interest = 0;
29        transactions = 0;
30        interestRate = 0;
31
32   }
33
34   public Banking(double int_balance, double int_interest){
35        balance = int_balance;
36        interest = 0;
37        transactions = 0;
38        interestRate = int_interest;
39
40   }
```

Figure 5: Sample of a student programming assignment.

```
4
5      // Instance variables declared here
6      private double balance;
7      private double interest;
8      private int transactions;
9      private static double interestRate = 0.0315;
10     private String acctName;
11     private String acctNumber;
12    // Static variable(interestRate) declared here
13  Scanner keyboard = new Scanner (System.in);
14     public void setAccount(double bal, double inter, int trans) //setting accoun
15     {
16         balance = bal;
17         interest = inter;
```

Figure 6: Sample of a student programming assignment.

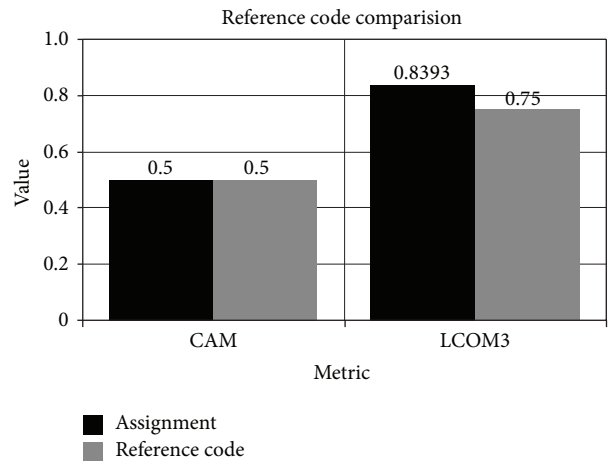

Figure 7: Comparison graph of assignments with reference code.

used in the classes methods. The CAM value ranges from 0 to 1. According to the standards, a CAM value closer to 1 (higher cohesion) is desirable. For example, if a class's CAM value is 0.5 we cannot say that the class is half cohesive and half uncohesive. In our study, we use the CAM value of the reference code as a standard for determining the quality of student assignments. We considered the CAM values of the Account class of all the assignments and the reference code for analysis as the Account class has more functionalities than the Main class where better cohesion is expected.

The reference code's Account class had a constructor to initialize the class attributes and other methods for performing the user's banking operations. On examining the sample from the student assignments, we found that 9% of the assignments had significantly lower values of CAM than the reference code. On close inspection we found that these assignments had unnecessary constructors and used method parameters that were not required. The grader can use the suggested feedback 2 to advise the student so that the code could be improved. Figure 5 is one such assignment that showed lower cohesion than the reference code.

4.2.4. LCOM3. The LCOM3 metric is an improved variation of the LCOM metric. It measures the cohesion of the class's methods in terms of the effective usage of the class's attributes. Hence, we will have more specific measurement over LCOM. It's value ranges from 0 to 2 and a value lesser than 1 is desired. In this study, we use the reference code's LCOM3 as a standard for analyzing the students assignments. We have considered the Account class from the student assignments and the reference code as it has more functionalities than the Main class.

We have found 27% of the student assignments to be having significantly higher values of LCOM3 than the reference code. On examining all such assignments we found that some assignments had unnecessary class attributes declared and others had not used the class attributes which were declared. So, the instructor can inspect these assignments for such problems and provide a feedback to the student using our guideline 3 and suggested feedback 3. Figure 6 is one such assignments that had higher LCOM3 value due to use of unrequired variables.

Figure 7 shows the comparison of CAM and LCOM3 metrics of a student assignment and the reference code.

Using software metrics to assess student programs one must consider the issues of metric dependencies. For example, the metrics for coupling and complexity can be compromised for each other. Code can be written to reduce complexity by the increasing modules which in turn increases coupling between the modules. Such metric dependencies should be considered while code inspection and giving feedback to the student.

We have also noticed smaller deviations among the metric values. Too smaller deviations from the reference code for RFC and CC values can be ruled out. As RFC count includes the method calls to the methods in the class libraries, these smaller variations could be the result of the calls to such methods in the assignment programs smaller variations could be a result of using or not using such functions in the student assignments. Similarly minor CC deviations can also be ignored as they are of less significance. But in case of LCOM3 and CAM, these smaller variations could be a result of an unnecessary method or an attribute and may be considered.

4.3. Survey Results. The grader was asked to find the quality issues related to cohesion and coupling complexity among the student assignments. Time was calculated for reviewing each assignment by the usual way and by using the approach of software metrics with reference code. Figure 8 is the timing graph for reviewing the assignments in the normal way without using our approach.

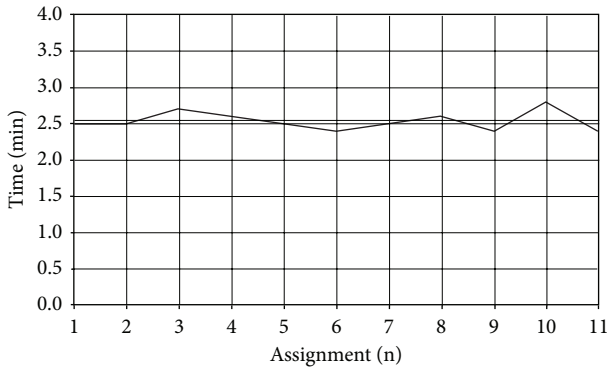

Figure 8: Inspection timing graph without proposed approach.

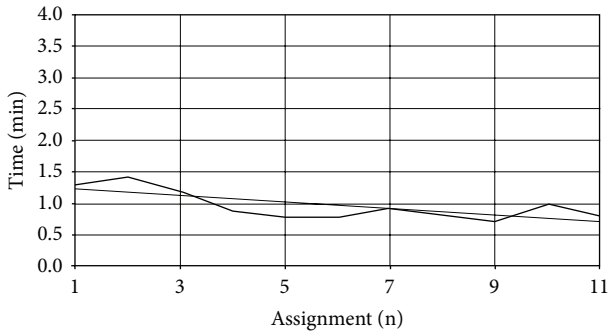

Figure 9: Inspection timing graph with proposed approach.

The times taken by the grader to inspect the assignments using our suggested approach are shown in Figure 9.

Comments. Using the approach of software metrics and reference code, the classes which had higher complexities and poor method relations were easily identified. In the normal way of reviewing the grader needed to start inspection from the Main class of the assignments to find out the quality issues. So, if there were design issues in the other class, time was wasted in reviewing the Main class which did not require review. Our approach helped the grader to find out that particular class of an assignment that required inspection.

The survey was also taken from the students after giving them the feedback. In this survey we will try to find out whether our approach helped the students. The students were asked to answer three questions regarding the feedback provided from the instructor. Figure 10 is the response from the students when the feedback is provided without the reference code.

64% of the students had difficulties in interpreting the feedback. Most of the students felt that more details were necessary to understand the feedback. So, we have provided them the reference code too and reviewed their responses. Figure 11 shows the survey from students after providing them with the feedback and reference code.

Now, most of the students understood the way in which they could improve their code after looking at their feedback and the reference code. Thus, the proposed approach can help the grader and student in reviewing the code quality

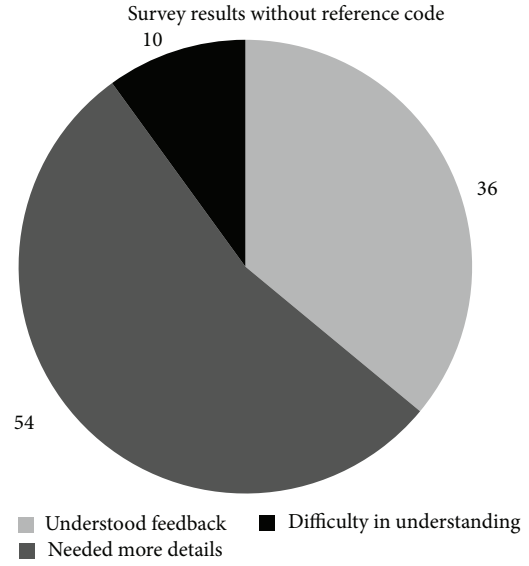

Figure 10: Survey results without giving out reference code.

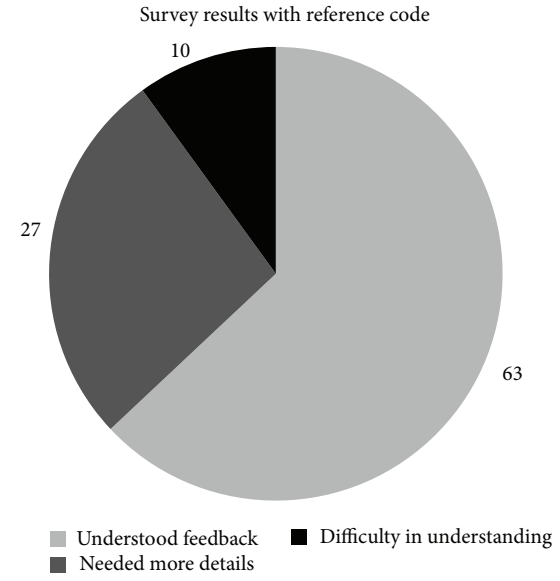

Figure 11: Survey results after giving out reference code.

5. Conclusion

The cohesion and coupling metrics, namely, CAM, LCOM3, and RFC, can help in assessing student programs which we have considered in our experiment. We have given guidelines for the grader in reviewing and providing possible feedback for the assignments when compared with the reference code's metrics. The reviewing times for the assignments reveal that our approach can help the instructor in identifying the classes that needs inspection and thereby saving time. The survey from students showed that providing the reference code along with the instructor's feedback can enhance their understandability of the provided feedback.

6. Future Work

The students sample that we have considered in our experiment is small, so we suggest that the experiment be conducted using a larger sample using a different assignment that has more than two classes. The Ca, Ce, and some other class metrics may be varied if larger assignments having more number of classes are used for the experiment. The metric values may be analyzed in a similar procedure and verified with our approach.

References

[1] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005.

[2] R. Cardell-Oliver, "How can software metrics help novice programmers?" in *Proceedings of the 13th Australasian Computing Education Conference (ACE '11)*, vol. 114, pp. 55–62, January 2011.

[3] S.-L. Hung, I.-F. Kwok, and R. Chan, "Automatic programming assessment," *Computers and Education*, vol. 20, no. 2, pp. 183–190, 1993.

[4] D. Jackson and M. Usher, "Grading student programs using ASSYST," *ACM SIGCSE Bulletin*, vol. 29, no. 1, pp. 335–339, 1997.

[5] F. Jurado, M. A. Redondo, and M. Ortega, "Using fuzzy logic applied to software metrics and test cases to assess programming assignments and give advice," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 695–712, 2012.

[6] S. A. Mengel and V. Yerramilli, "A case study of the static analysis of the quality of novice student programs," *ACM SIGCSE Bulletin*, vol. 31, no. 1, pp. 78–82, 1999.

[7] T. Wang, X. Su, Y. Wang, and P. Ma, "Semantic similarity-based grading of student programs," *Information and Software Technology*, vol. 49, no. 2, pp. 99–107, 2007.

[8] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Computers and Education*, vol. 41, no. 2, pp. 121–131, 2003.

[9] M. Maxim and A. Venugopal, "FrontDesk: an enterprise class web-based software system for programming assignment submission, feedback dissemination, and grading automation," in *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '04)*, pp. 331–335, September 2004.

[10] M. Wick, D. Stevenson, and P. Wagner, "Using testing and JUnit across the curriculum," *ACM SIGCSE Bulletin*, vol. 37, no. 1, pp. 236–240, 2005.

[11] B. J. Bowman and W. A. Newman, "Software metrics as a programming training tool," *The Journal of Systems and Software*, vol. 13, no. 2, pp. 139–147, 1990.

[12] T. J. McCabe, "A complexity measures," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[13] R. J. Leach, "Using metrics to evaluate student programs," *ACM SIGCSE Bulletin*, vol. 27, no. 2, pp. 41–43, 1995.

[14] S. R. Chidamber and C. F. Kemerer, "Metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[15] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/down.html.