DO CLASS COMMENTS AID JAVA PROGRAM UNDERSTANDING?

Eriko Nurvitadhi[1], Wing Wah Leung[2], and Curtis Cook[3]

Abstract *- This paper describes an experiment that investigates the effects of class and method comments on Java program understanding among beginning programmers. Each of the 103 students from CS1 class at Oregon State University was given one of four versions (No comments, only method comments, only class comments, and both method and class comments) of a Java database program and answered questions about the program. The results indicated that method comments do increase low-level program understanding, while class comments did not increase high-level understanding. This raises questions about the role of class comments in Object-Oriented programs, as well as the kind of commenting guidelines that should be used in teaching CS1 classes.*

Index Terms – class vs. method comments, Java program understanding, empirical study.

INTRODUCTION

Making a program understandable is critical as it plays an important role in most common programming tasks such as maintenance, testing, and debugging. As much as half of the time spent on these tasks is devoted to understanding the program. Studies of the effects of comments on program comprehension have shown that effective use of comments can significantly increase program comprehension [7,9,11]. However, these studies focus primarily on imperative or non-OO programs.

An OO program can be described as a community of objects that work together to complete the same objective [3]. A class is a blueprint of an object [6], describing the states and behaviors of the object. A behavior is, in practice a method, e.g. a function or procedure. Since, an OO program is comprised of classes, it is therefore important for a programmer to grasp the meaning of these classes and the methods in these classes.

The more knowledge the programmer has about the classes and methods within the program, the more understanding he/she has about the program. Moreover, a class may have many behaviors that sometimes are not reflected in the design of the class. Therefore, more detailed understanding of these behaviors may be necessary for the programmer to understand the program.

This suggests that OO program commenting should include both a class-based comment that provides an overview of a class and a method-based comment that gives information about method behaviors. More specifically, a class-based comment should be helpful in developing a high-level knowledge of a program, such as the purpose of the class, what the class does, or the interconnection between classes. On the other hand, a method-based comment should provide more low-level understanding of the program, such as the purpose of the method, implementation technique, and inner workings of a class. Examples of class and method-based comments are given in Figure 1.

```
//=========================================
// This class acts as a product database.
// Products can be added and deleted.
// Information on the currently stored
// products can also be printed out.
//=========================================
class StoreDB {
       // The code for StoreDB
}
```

FIGURE 1A
A CLASS-BASED COMMENT.

```
// =========================================
// This function adds a product p to the
// database. The function stores the product
// by looping through the storage array and
// putting the product in the first null
// slot it found. If the storage is full,
// then an error message is printed out.
// =========================================
public void Add(Product p){
       // The code for Add(p)
}
```

FIGURE 1B
A METHOD-BASED COMMENT.

All of this raises questions about recommendations for when to use and guidelines for what to include in class and method comments. In particular, we are interested in what recommendations and guidelines for commenting should be taught and used in beginning Java programming courses. Java books may provide some general guidelines for commenting; in most instances they do not provide detailed information about what should be included in the various types of commenting. However, these textbooks have differing recommendations and guidelines for class and method commenting. Some books provide guidance for both types of comments, while others provide guidance for only one of them. In the discussion below we present a sample of the differing recommendations and guidelines.

[1] Eriko Nurvitadhi, Oregon State University, Corvallis, OR 97330 nurviter@engr.orst.edu
[2] Wing Wah Leung, Oregon State University, Corvallis, OR 97330 leungwi@engr.orst.edu
[3] Curtis Cook, Oregon State University, Corvallis, OR 97330 cook@engr.orst.edu

0-7803-7444-4/03/$17.00 © 2003 IEEE **November 5-8, 2003, Boulder, CO**
33rd ASEE/IEEE Frontiers in Education Conference

Arnow and Weiss [1] recommend that, "there should be comment before the line that starts with the keyword class, indicating the purpose and the behavior of the code." This statement shows strong support for the class-based comment. However, the book does not provide any recommendation in regards of commenting style for a method. Wu [12] suggests using header comments. Header comments here means the comments located at the beginning of the code that describe a program. The book also says, "We may also include header comments at the beginning of methods to describe their purpose." Later the book provides a template for method headers that includes the information about the name, purpose, available parameters and return value of the method. Horstmann [5] mentions that currently there is no universal standard for method comments. Nevertheless, the book insisted that every method, except "main", must have comments that describe its purpose. It also encourages the use of header comment in the beginning of a file to explain the purpose of the file. Holmes [4] presents a template for a typical Java program, which includes a header comment at the beginning of the program explaining the name and purpose of the program. Finally Wang [10] recommends, "Precede each class with appropriate doc comments."

Thus, most of these books provide information about the syntax or templates for commenting, but offer little guidance about what should be included in these comments. This paper investigates commenting styles for Java programs. We conducted an empirical study of the effects of class and method commenting on program understanding using subjects from a CS1 class. The results of this study are reported in this paper.

The rest of the paper is organized as follows: Section 2 describes our experiment. Section 3 presents the results of the experiment. Section 4 discusses the results, as well as providing guidelines for class and method commenting.

EXPERIMENT

Hypotheses

We formulated four hypotheses in accordance to our previous discussion about class and method comments:

H(1): There will be no difference in high-level program understanding between a Java program with class-based comments and without class-based comments.

H(2): There will be no difference in low-level program understanding between a Java program with method-based comments and without method-based comments.

H(3): There will be no difference in program understanding between a Java program with both class-based and method-based comments and a program with just one of these types of commenting or no commenting.

H(4): There will be no difference in subject rating of the difficulty of program understanding among Java programs with both class-based and method-based commenting, with just one of these types of commenting, and with no commenting.

Independent and Dependent Variables

We define two independent variables, class-based and method-based comments. These comments are constructed based on the guidelines from the textbooks we examined, as well as what is recommended by Javadoc [2]. The class comments (Figure 1A) contain the description of the class, what it can do, what kind of information it has, and its relationship with other classes. The method comments (Figure 1B) provide the description of the method, its pre-condition, such as the description of the parameters, and its post-condition, such as any side effects. Additionally, it may also include the internal behavior of the method. Four different versions of the same program were used to accommodate each combination of these independent variables:

None: No class-based or method-based comments
Class: Class-based comments only
Method: Method-based comments only
Both: Both class-based and method-based comments

A high-level question

What does the Store object do?

a. Instantiates StoreDB object. Adds products to the StoreDB object. Displays the content of the StoreDB.
b. **Instantiates StoreDB object. Adds products to the StoreDB object. Deletes products from the StoreDB object. Displays the content of the StoreDB.**
c. Instantiates StoreDB object. Adds products to the StoreDB object.
d. Instantiates StoreDB object. Adds products to the StoreDB object. Finds products from the StoreDB object.
e. Instantiates StoreDB object. Adds products to the StoreDB object. Deletes products from the StoreDB object. Finds products from the StoreDB object. Displays the content of the StoreDB.

A low-level question

How many maximum products can be deleted by the "Delete" function in the "StoreDB" class?

a. 1
b. Depends on the current value of "ItemInStorage"
c. Depends on the current value of "capacity"
d. Depends on the length of the array "Storage"
e. Both c and d are correct

FIGURE 2
EXAMPLE OF A HIGH-LEVEL AND A LOW-LEVEL QUESTION. THE CORRECT RESPONSE IS GIVEN IN BOLDFACE.

Program understanding was measured by a 10-question comprehension quiz. The 10 questions were divided into 5 high-level questions and 5 low-level questions. The high-level questions were designed to assess the subject's high-level understanding of the program, such as what the program does, or how one class relates to another. The other 5 questions assessed subject low-level understanding, such as how a certain method would behave, or how a particular behavior is implemented. Figure 2 shows an example of a high-level and a low-level question.

We also asked the subjects to rate the difficulty in understanding the program using the 7 point scale: 1 for very difficult, 2 for difficult, 3 for moderately difficult, 4 for moderate, 5 for moderately easy, 6 for easy, and 7 for very easy.

Subjects and Materials

Subjects for this study were students in the second quarter of a three-quarter introductory computer science sequence. This class taught OO concepts and used Java for the programming assignments. The comprehension test was given in the ninth week of the quarter. By the time students took the test, they had already learned all of the syntax for the OO structures that appeared in the program used in the experiment. Moreover, they had already written several programs beyond the complexity of the tested program.

A total of 103 students participated in our experiment. Subjects completed a background questionnaire that asked about their GPA and Java programming experience. See Table 1. Analysis of the background data showed no significant differences between the groups.

TABLE I
SUBJECT BACKGROUND

	GPA	Java Experience (Months)
None	3.26	8.50
Class	3.30	8.96
Method	3.41	7.12
Both	3.32	6.69

The study was conducted during four of the course recitation sessions. Four versions of the programs were handed out evenly among these subjects in each session. All subjects had access to the program during the comprehension quiz. The program used in the experiment is the so-called Store database program. It is a fairly simple Java program, but includes the main OO concepts, such as inheritance, overriding, and overloading. It creates a storeDB object that can store maximum of 10 products. It then adds some products to the database, deletes two of them, and then adds three more products again. Finally, the content of the database is printed out in a sorted order. The syntactical structures presented in the program include IF-THEN-ELSEs, FOR loops, and ARRAYs. We use meaningful identifier names in the program. Moreover, no inline comments were used. Copies of the program, comprehension quiz, and other materials are available from the authors.

RESULTS

The comprehension quiz was scored one point per question. From Table 2 we see that, as expected, the None group performed worse and the Both better than the other groups. An ANOVA test found significant differences between the four groups for Total score ($F=3.583$, $df=3$, $p=0.0165$) and Low-level score ($F=4.416$, $df=3$, $p=0.0059$). Fisher's PLSD found significant differences between Both and None and between Method and None for both Total score ($p=0.0025$, $p=0.0175$) and Low-level score ($p=0.0009$, $p=0.0079$). As a check, Mann-Whitney nonparametric tests confirmed these differences. Thus we can reject hypothesis H2 for no comments but not for class-based comments. Also we cannot reject hypotheses H1 and H3.

TABLE 2
MEANS FOR THE COMPREHENSION QUIZ SCORES AND THE UNDERSTANDING RATINGS AMONG THE FOUR GROUPS

	Low-level	High-level	Total	Understanding Rating
None	2.28	2.64	4.92	3.56
Class	2.84	2.84	5.69	3.65
Method	3.16	2.92	6.08	3.74
Both	3.38	3.0	6.38	3.61

The two major surprises in these results were how well the Method group performed and how poorly the Class group performed. We expected the Method group to perform better than the Class group on the Low-level questions but not on the High-level questions. Further we expected the Class group to perform much better than the None group, especially on the High-level questions. But this was not the case.

Looking at the combined effect of the two types of comments, we did not see any significant differences in overall program comprehension. This seems to indicate that class-based comments did not help high-level program understanding as much as we thought it would.

Additionally, we analyzed the performance of the four groups on each question. See Figure 3. There is little difference in performance for most of the questions. However, Chi-square test revealed significant differences between the Method and None groups for question 6 ($p=0.0377$), and between the Both and None groups for question 8 ($p=0.0254$). Both questions 6 and 8 ask detailed questions about method behavior. Question 6 asks about possible erroneous situation for a method in the program. The method-based comment contains a description about how the method worked and was implemented. Thus, if the subject understood the comment, he/she should be able to answer the question correctly. A similar situation applied for question 8, which also asks about possible erroneous case for another method.

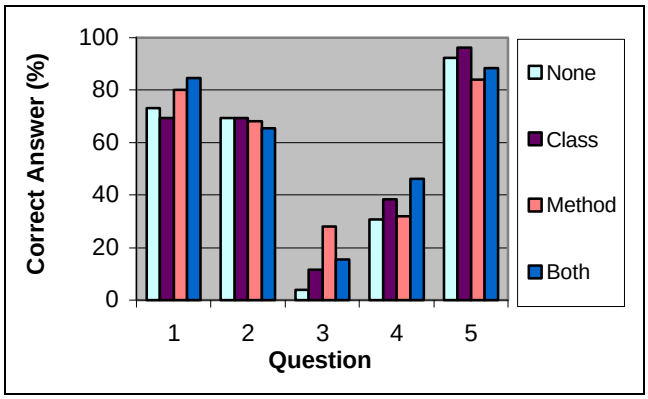

FIGURE 3A
PERCENTAGE OF CORRECT ANSWER FOR EACH HIGH LEVEL QUESTION (QUESTION 1 THROUGH 5).

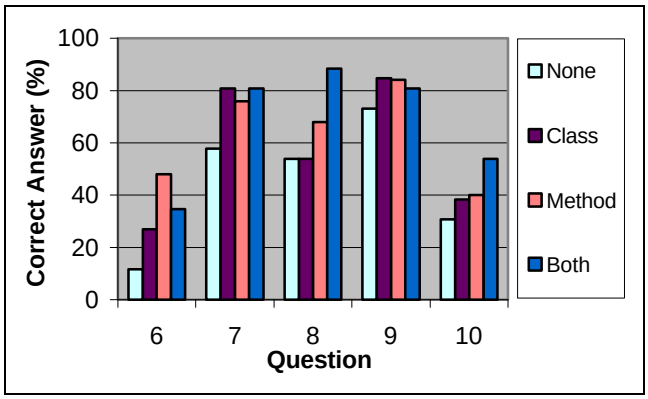

FIGURE 3B
PERCENTAGE OF CORRECT ANSWER FOR EACH LOW LEVEL QUESTION (QUESTION 6 THROUGH 10).

The subjects also rated on a 1-7 scale (1=very difficult, 7=very easy) the difficulty of understanding the program. From Table 2 we see that there is little difference in the ratings among the four program groups and the ranges indicate that the subjects think the program was between moderate and moderately difficult to understand. This may be due to the simple design of the program and the use of the meaningful identifier names.

CONCLUSION

Our experiment found that method-based comments impacted program understanding more than class-based comments. One possible reason why class-based commenting had little impact was that the Java database program was small and fairly simple. It consisted of 5 classes and 17 methods with a total of 172 lines of actual codes, excluding the 27 lines of class comments and 70 lines of method comments. Therefore, the relationships between classes, as well as the purpose of the classes might have been obvious and already revealed by the simple design of the program. On the other hand, real world programs used in industry are substantially larger and more complicated. In such cases, class-based commenting is highly likely to impact program understanding since there are more classes and complex interconnections and relationships among the classes.

This experiment suggests the need to provide students in beginning OO programming courses with clearer and more detailed guidelines for the commenting type and style. Students at this level are typically unaware of the importance of program understanding. Hence explicitly mentioning that making a program understandable is the primary reason for commenting would assist beginning programmers in constructing their comments.

So what commenting guidelines should be offered? Several schemes for documenting object-oriented programs have been proposed. One example [8] suggests applying the object-oriented technology to the documentation since one major role for documentation is enhancing reuse. This scheme concentrates on describing interfaces and implementation aspects that facilitate reuse and maintenance. It provides external and internal views of both the static and dynamic aspects of software components. Then inheritance is applied by reusing and extending existing documentation and using a control mechanism to enforce information hiding. Such a scheme is appropriate for industrial-sized software, but seems excessive for the typical small program assignments given in introductory programming classes.

```
// Class Name
// The purpose of the class
// Relationships with other classes (if any)
// Descriptions of public members of the class (if any)
// Implementation notes (if any)
Class {
      // Code for the class
}
```

FIGURE 4A
CLASS-BASED COMMENT TEMPLATE.

```
// The purpose of the method
// Descriptions of the parameters (if any)
// Return value (if any)
// Implementation notes (if any)
Public void myMethod (int arg){
      // Code for the method
}
```

FIGURE 4B
METHOD-BASED COMMENT TEMPLATE.

Our experiment suggested the need for guidelines concerning methods and class comments. Based on commenting guidelines offered in the textbooks we

examined, we created templates for both class-based and method-based commenting. See Figure 4. We feel that these guidelines are more appropriate for commenting introductory courses taken by beginning programmers.

As future work, we encourage the replication of our experiment and further investigation of the impact of method-based and class-based commenting on program comprehension. Further, it would be interesting to determine the effect of these types of comments on the comprehension of larger or industrial-sized programs. Finally, we intend to investigate the effectiveness of the templates presented in Figure 4.

ACKNOWLEDGMENT

The authors would like to thank Chris Wallace for her help that made it possible to conduct this experiment. The authors would also like to thank the anonymous reviewers for their comments that clarified the presentation of the results.

REFERENCES

[1] Arnow, D., Weiss, G., "Introduction to Programming Using Java, an Object-Oriented Approach", *Addison-Wesley* 1998.

[2] Bloch, J., "Effective Java Programming Language Guide", *Addison-Wesley* 2001.

[3] Budd, T.A., "An Introduction to Object-Oriented Programming", *Addison-Wesley*, 3rd edition, 2002.

[4] Holmes, B., "Programming with Java", *Jones and Bartlett* 1998.

[5] Horstmann, C., "Computing Concepts with Java Essentials", *John Wiley & Sons, Inc* 1998.

[6] Lewis, J., Loftus, W., "Java Software Solutions: Foundations of Program Design", *Addison-Wesley* 1997.

[7] Oman, P. and Cook, C., "Typographic Style is More Than Cosmetic", *CACM 33* (May 1990), 506-520.

[8] Sametinger, J., "Object-Oriented Documentation", *Journal of Computer Documentation 18* (1994), 3-14.

[9] Takang, A.A., Grubb, P.A., Macredie, R.D., "The Effects of Comments and Identifier Names on Program Comprehensibility: An Experimental Investigation", *Journal of Programming Languages 4* (1996), 143-167.

[10] Wang, P.S., "Java with Object-Oriented Programming and World Wide Web Applications", *PWS Publishing* 1999.

[11] Woodfield, S.N., Dunsmore, H.E., Shen, V.Y., "Effect of Modularization and Comments on Program Comprehension", *In Proceedings, 5th International Conference on Software Engineering* (March 1981), IEEE Computer Society Press, 215-223.

[12] Wu, C.T., "An Introduction to Object-Oriented Programming with Java", *McGraw-Hill* 1999.