THE ADHERENCE OF OPEN SOURCE JAVA PROGRAMMERS TO STANDARD CODING PRACTICES

Mahmoud O. Elish
Department of Computer Science
George Mason University
Fairfax VA 22030-4400 USA
melish@gmu.edu

Jeff Offutt[†]
Information and Software Engineering
George Mason University
Fairfax VA 22030-4400 USA
www.ise.gmu.edu/~ofut/
ofut@ise.gmu.edu

ABSTRACT
The use of agreed-upon coding practices is believed to enhance program comprehension, which directly affects reuse and maintainability. This paper describes a controlled small-scale experiment that tries to determine how well open source Java programmers adhere to a set of well publicized coding practices. The experiment evaluated 100 arbitrarily selected open source Java classes from different programmers with respect to 16 standard coding practices. The results of this experiment indicate that open source Java programmers do not always adhere to standard coding practices. It was found that only 4% of the subject classes have no violations to any of the 16 standard coding practices and there were only 5 of 16 coding practices that all subjects followed. It was also found that there are positive correlations between the number of violations found in a class and its lines-of-code, number of methods, and number of attributes.

KEY WORDS
Coding practices, experiment, Java classes, open source software, software quality.

1. INTRODUCTION

The availability and use of open source software has increased recently as they give users the freedom to run, distribute, study, change and improve them [1]. However, these open source software applications cannot be useful unless they are of high quality and are perceived to be of high quality. An important indicator of software quality is that of standard coding practices. Whether programmers adhere to coding standards is optional in the sense that programs that violate the standards can still be syntactically and functionally correct. Despite this, adherence to standard coding practices is considered to be essential to ensure readable and understandable software, and thus more likely to be easy to maintain and reuse [2, 3, 4]. As one coding standards author said: "In order to write great software, you have to write software greatly" [5].

In this study, a controlled small-scale experiment was conducted to determine how well Java programmers who develop software with the open source model adhere to standard coding practices. One hundred arbitrarily selected open source Java classes were evaluated with respect to sixteen standard coding practices. This paper reports on the experiment and its results.

The next section describes the standard coding practices that were used in this experiment. Section 3 states the hypothesis and the corresponding research questions. The experimental design, including its subjects, measurements, and data analysis are presented in Section 4. A discussion and several directions for future work are given in Section 5.

2. STANDARD CODING PRACTICES

One of the difficulties with measuring adherence to standard coding practices is that there are many styles in use. In fact, most software engineers would agree that adhering to some style is the most important step; the particular style followed is often secondary. This makes measuring adherence to programming standards a potentially expensive task. Fortunately, a number of standard coding practices for the Java programming language have been developed and are fairly widely used by the Java development community [2, 3, 4]. This experiment uses sixteen standard coding practices that satisfy two criteria: (1) they are widely used, and (2) they can be measured by automated tools. In particular, we were able to automatically collect data using the two Java analysis tools JStyle and RSM [6, 7]. The sixteen practices are grouped into four categories: naming conventions (N), formatting (F), control structure use (C), and visibility (V). The following subsections define these sixteen coding practices, in some cases with rationales.

[†] *Partially supported by the U.S. National Science Foundation under grant CCR-00-97056.*

2.1. Naming Conventions

Naming conventions provide guidance for how to choose names for Java's classes, attributes, methods, constant, and interfaces. The intent is that the category of the name should be apparent from the spelling or punctuation of the name.

- N1 – Class names should be in mixed case starting with uppercase, and with the first letter of each internal word capitalized [3, 6].
- N2 – Attribute names should be in mixed case starting with lowercase, and with the first letter of each internal word capitalized [3, 4, 6].
- N3 – Method names should be in mixed case starting with lowercase, and with the first letter of each internal word capitalized [3, 4, 6].
- N4 – Constant (final attribute) names should be all uppercase using underscore to separate words [3, 4, 6].
- N5 – Method names should be different from the class name in which they are defined [6]. *Rationale: Since a method has a return type, it will not be treated as a constructor. Therefore, it should be named differently to avoid confusion.*

2.2. Formatting

This experiment considers the following standard coding practices associated with file formatting and organization.

- F1 – Line length should not exceed 80 characters [3, 4, 7]. *Rationale: Lines with more than 80 characters length may not be handled well by many terminals and tools, thus resulting in line truncation, horizontal scrolling, and wrapping, both of which make the code difficult to read.*
- F2 – TAB characters should be avoided [4, 7]. *Rationale: Using TAB characters within source code may result in a file that is print or display device dependent.*

2.3. Control Structure Use

This experiment examines the following standard coding practices related to control structure use.

- C1 – The 'switch' statement should have a 'default' condition [7]. *Rationale: Having a 'default' condition in a 'switch' statement makes it deterministic.*
- C2 – The keyword 'break' should not be used outside a 'switch' block [4, 7]. *Rationale: Using the 'break' statement outside a 'switch' block disrupts the linear flow of logic in a program.*
- C3 – The 'continue' statement should be avoided [4, 7]. *Rationale: Using the 'continue' statement breaks the linear flow of logic in a program.*
- C4 – Assignments should be avoided in 'while', 'do … while', 'for', and 'if' logical conditions [6, 7]. *Rationale: Using '=' instead of '==' is likely to be a typo. If not, it should be avoided as it makes the code difficult to understand.*

2.4. Visibility

This experiment focuses on the following standard coding practices associated with the visibility of class data members.

- V1 – Class attributes should not be declared public [3, 4, 6, 7]. *Rationale: Public attributes break data encapsulation and information hiding concepts.*
- V2 – The constructor in a nonpublic class should not be public [6]. *Rationale: A nonpublic class cannot be instantiated outside the package in which it is defined, and hence there is no need for a public constructor for such a class.*
- V3 – A public class should have at least one public member or protected constructor [6]. *Rationale: A class cannot be used outside the package unless it has public constructor(s) and/or public member(s).*
- V4 – Each local variable in a method should be used at least once [6]. *Rationale: Unused local variables in a method reduce the cohesiveness of the method.*
- V5 – Each parameter in a method should be used at least once [6]. *Rationale: Unused parameters in a method reduce the cohesiveness of the method.*

3. HYPOTHESIS AND RESEARCH QUESTIONS

The hypothesis of this study is that *open source Java software classes do not always conform to all standard coding practices described above*. In other words, there are violations of some of the above coding practices in a high percentage of Java classes. Given this hypothesis, the following four research questions are investigated:

(1) Is there a correlation between class size, measured by Lines-Of-Code (LOC), and the number of violations found in that class?

(2) Is there a correlation between the number of methods (NOM) defined in a class and the number of violations found in that class?

(3) Is there a correlation between the number of class attributes (NOA) and the number of violations found in that class?

(4) Is there a correlation between the comment density of a class (i.e. number of comment lines / total number of lines) and the number of violations found in that class?

4. THE EXPERIMENT

A controlled small-scale experiment was conducted to determine how well open source Java classes conform to the standard coding practices defined in Section 2. The details of this experiment are discussed in the following subsections.

4.1. Subjects

The subjects of this experiment are 100 arbitrarily selected open source Java classes that were collected from different open source web sites [8, 9, 10, 11]. Two factors that could affect the results of this experiment are the author of the classes and the size of the programs. The program author could affect the results because it is probably that individual programmers would use the same coding style in all of their classes. This variable was controlled by classes that were all written by different programmers. The sizes of the program that the classes appear in have a lesser impact. In OO software, each class is relatively independent and the heavy reliance on reuse dictates that the program size may vary anyway. This variable was controlled by focusing on individual classes, irrespective of program. At the class level, it is likely that the sizes of classes in terms of LOC, NOM, and NOA may vary significantly and impact the adherence to standard coding practices, thus this was used as an experimental variable and we measure correlations between standards violations and class size.

4.2. Measurements and Data Analysis

Two tools were used to analyze the collected Java classes: JStyle version 4.6 [6], and RSM (Resource Standard Metrics) version 6.03 [7]. Both tools have free demo versions and are available online. Table 1 summarizes the basic descriptive statistics for the 100 classes. The number of violations is the total number of distinct violations of the 16 standard coding practices found in a class. As indicated in Table 1, the mean number of violations found per class is 2.6. The number of violations found per class varies from 0 to 8, with median and mode values of 3.

Figure 1 illustrates a histogram that shows the number of classes that violates each one of the 16 standard coding practices described in Section 2. As shown in this histogram, all subject classes follow the coding standards N1, N5, C1, V2, and V3, and all other standards were violated at least once. That is, only 5 out of the 16 standard coding practices are followed by all subjects.

4.3. Discussion

There are several aspects of these data that warrant further examination, particularly for the practices that were violated the most often. The violations of the naming practices can be misleading. Although there is fairly wide agreement among Java programmers, naming practices are somewhat arbitrary, and it is more important that a programmer follow **some** practice than he or she follow **a specific** practice. It is entirely possible that some of the reported violations were actually cases in which the programmer simply followed a different practice. Unfortunately, it is impossible to check for a variety of unknown practices, thus we accept these data with the caveat that adherence to naming standards might in fact be better than it appears.

Table 1. Descriptive statistics for the 100 classes

	LOC	NOM	NOA	Comment Density	No. of Violations
Mean	269.8	10.0	20.9	15.2%	2.6
Standard Dev.	323.2	7.1	18.6	12.2%	1.4
Mode	107.0	8.0	3.0	0.0%	3.0
Minimum	18.0	1.0	0.0	0.0%	0.0
1st Quartile	108.8	6.0	8.0	6.9%	1.8
Median	179.0	8.0	18.0	12.3%	3.0
3rd Quartile	280.5	12.0	25.5	21.3%	3.0
Maximum	1880.0	50.0	109.0	72.4%	8.0

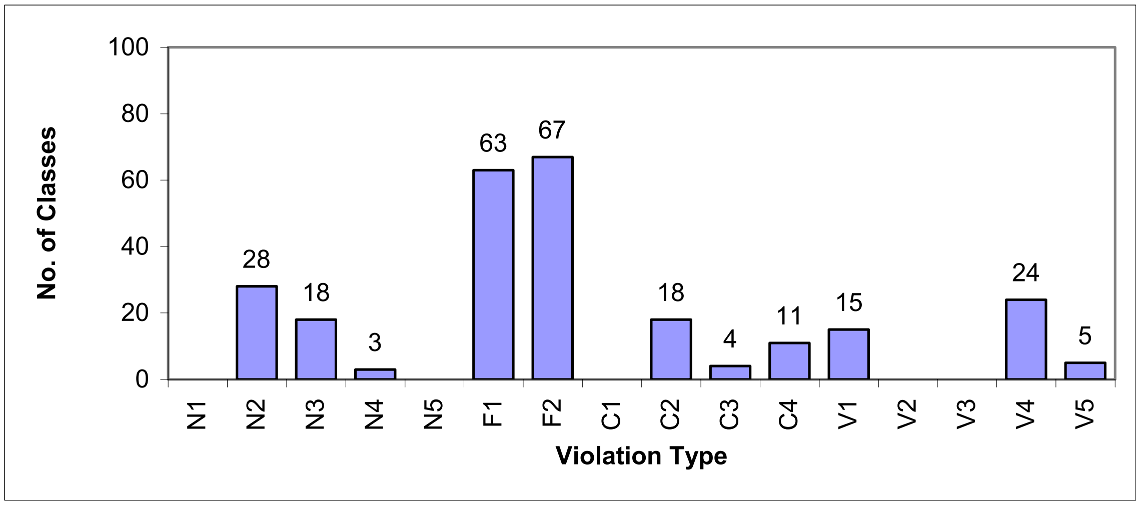

Figure 1. Violation type vs. number of classes

More disturbing is the fact that two formatting practices are violated by most (over 60%) of the subject classes. This indicates that the programmers of these classes do not pay much attention to F1 and F2 coding practices. We have several potential explanations for these data. One might be that programmers simply do not pay much attention to formatting, perhaps because it is less visible to the naked eye. Another explanation might reflect a limitation of the editing tools; by default, *vi* inserts tabs automatically in text, and most mouse-based editors completely screen the programmers from tabs and line-wrapping. On the other hand, reuse and the open source software development model dictates that different developers will use different editors and software development environments. Over time, violations of these standards will lead to confusion and faults in the software.

A detailed examination of the C2 practice violations reveals that some programmers use the "break" statement to force an early exit from a loop, and others to implement a loop with a condition to exit in the middle of the loop body. Although indiscriminate use of the "break" statement is certainly undesirable, many experienced programmers would agree that these uses of the "break" statement are acceptable. It is very hard to distinguish between "acceptable" violations of C2 and "unacceptable" violations.

The number of violations of the visibility practices surprised us. V1 prohibits public class attributes, and while there are certainly times when it is necessary, the fact that 15% of the Java classes we looked at used public class attributes is disturbing. It is quite possible that this documents a significant problem with open source software. This is particularly true when added to a recent result by Schach, Offutt et al. [12], which found that the open source Linux kernel exhibits so much common coupling (public class attributes) that it could become un-maintainable in the future.

The violations of V4 and V5 might not be such a cause for alarm. Open source programs are frequently works in progress and many methods may contain variables and parameters that were introduced as placeholders for expected future use and thus not currently being used. On the other hand, unused variables and parameters may also be a result of careless programming or poor use of inheritance. We know of no automated way to distinguish these cases.

Table 2 provides the percentage of classes that completely followed all practices in each of the four standard coding practice categories considered by this experiment. On the positive side, over 50% of the subject classes follow the naming conventions, control structure use, and visibility standards. On the negative side, the category that was violated the most is formatting, where only 12% of the subject classes follow both F1 and F2 coding practices. This indicates potential for a long-term problem.

Table 2. Percentage of classes that follow each coding practice category

Coding practice category	% of classes that follow this category
Naming conventions (N)	60%
Formatting (F)	12%
Control structure use (C)	72%
Visibility (V)	63%

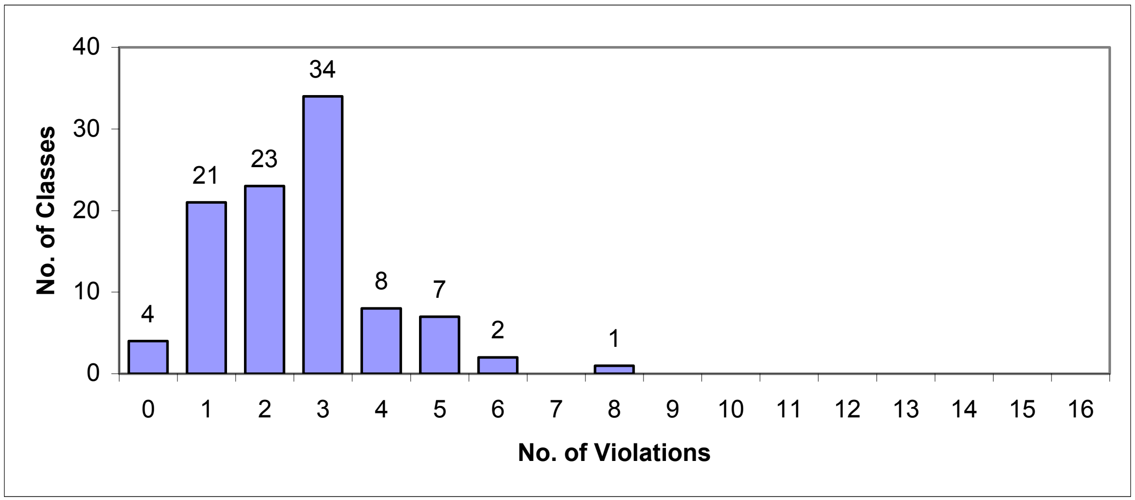

Figure 2. Number of violations vs. number of classes

The histogram in Figure 2 plots the classes that have the same number of violations. Only 4% of the subject classes have no violations, that is, follow all the 16 standard coding practices, which is disturbing. On the other hand, the distribution in Figure 2 peaks early, with 90% of the subject classes having at most four violations.

4.4. Effect of Class Size on Violations

A related question that we had is whether the class size would have an impact on the number of coding standards violations. Correlation analyses were performed to test for correlations between the numbers of violations found in classes and their sizes in terms of LOC, NOM, NOA, and comment density. At the 0.05 level of significance (95% confidence level), the correlation analyses concluded that there do exist positive correlations between the number of violations found in a class and its LOC, NOM, and NOA. However, it was found that there is no correlation between the number of violations found in a class and its comment density. Figures 3, 4, 5, and 6 show the scatter plots with lines of regression of the number of violations found versus LOC, NOM, NOA, and comment density.

These figures indicate that as the number of lines, methods, and attributes increase, the number of coding standards violations also increase. It is encouraging that the increases in violations are linear with the increases in size, that is, the frequency of violations does not change.

4.5. Summary of Results

The primary results and observations of this experiment can be summarized as follows.

- The mean number of violations found per class is 2.6.
- The number of violations found per class varies from 0 to 8.
- All subject classes follow N1, N5, C1, V2, and V3 coding practices, that is, only 5 out of the 16 standard coding practices are followed by all subjects.
- 60% of the subjects follow naming convention practices; 12% follow formatting practices; 72% follow control structure use practices; and 63% follow visibility practices
- Only 4% of the subject classes have no violations, i.e. follow all the 16 standard coding practices.
- 90% of the subject classes have at most 4 violations.
- There are positive correlations between LOC, NOM, and NOA of a class and the number of violations found in that class.
- There is no correlation between the comment density of a class and the number of violations found in that class.

5. CONCLUSIONS

This paper has described an experiment conducted to assess the quality of open source Java programmers in terms of their adherence to 16 standard coding practices. Based on the experimental results, the hypothesis was accepted, that is open source Java software does not always conform to the 16 standard coding practices examined by this experiment.

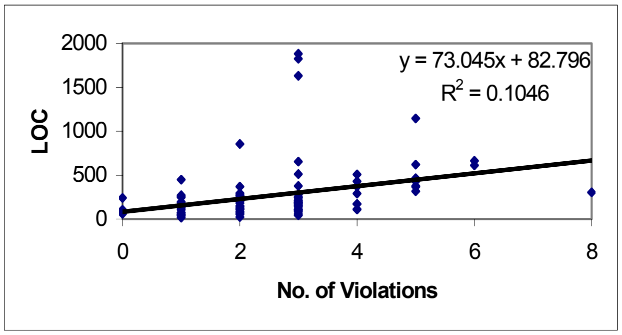

Figure 3. Scatter plot of number of violations vs. LOC

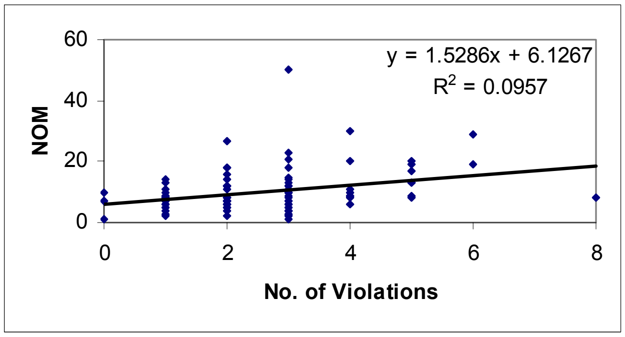

Figure 4. Scatter plot of number of violations vs. NOM

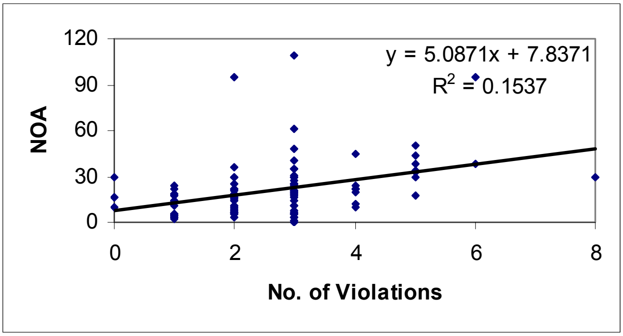

Figure 5. Scatter plot of number of violations vs. NOA

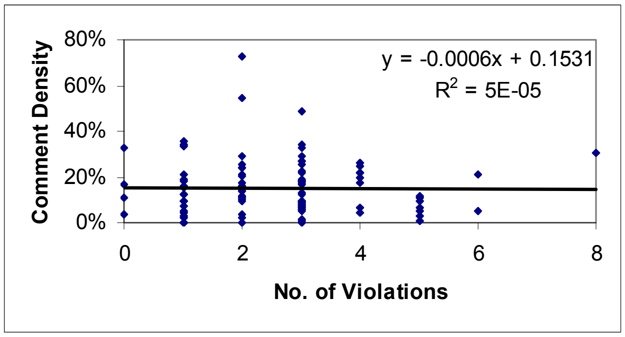

Figure 6. Scatter plot of number of violations vs. comment density

However, the data give many reasons to be positive. Many of the violations are on standard coding practices that are in some sense arbitrary (the naming standards). Also, the fact that 90% of the classes violate at most four of the standard practices is a positive sign. The most troubling of the standards violations are the formatting (F1 and F2) and the use of public attributes (V1). These can potentially lead to severe problems, especially when we consider that open source software needs to be easily read and understand by programmers at different organizations who use different tools.

This study evaluated single Java classes with approximately 270 LOC on average. An interesting question that remains is how open source software compares to closed source software. Answering such a question would be complicated because it seems likely that there would be wide variance across development organizations.

REFERENCES

[1] W. Harrison, Editorial: Open Source and Empirical Software Engineering, *Empirical Software Engineering Journal*, 6, 2001, 193-194.

[2] S. Ambler, *Writing Robust Java Code: The AmbySoft Inc. Coding Standards for Java*, Jan. 2000. Online at http://www.AmbySoft.com/javaCodingStandards.pdf

[3] Sun Microsystems, *Code Conventions for the Java Programming Language*, Revised Apr. 1999. Online at http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

[4] Geotechnical Software Services, *Java Programming Style Guidelines*, Version 3.0, Jan. 2002. Online at http://www.geosoft.no/javastyle.html

[5] C. Badeaux, *Netscape's Software Coding Standards Guide for Java*, Online at http://developer.netscape.com/docs/technote/java/codestyle.html

[6] Man Machine Systems: JStyle. Online at http://www.mmsindia.com/jstyle.html

[7] Resource Standard Metrics. Online at http://m2tech.net/rsm/

[8] Code Beach – Free and Open Source Code and Tutorials. Online at http://www.codebeach.com/

[9] Free Code. Online at http://www.freecode.com/index/

[10] Java Boutique. Online at http://www.javaboutique.internet.com/javasource/

[11] Java's Programmer Source Book. Online at http://www.codeguru.com/java/

[12] S. R. Schach, B. Jin, D. Wright, G. Heller and J. Offutt, Maintainability of the Linux Kernel, *IEE Proceedings Journal: Special Issue on Open Source Software Engineering*, 2002 (to appear).