# An Empirical Comparison of Modularity of Procedural and Object-oriented Software


Lisa K. Ferrett
AT&T Government Solutions, Inc.
1900 Gallows Road
Vienna, VA 22182 USA
ferrett@att.com

Jeff Offutt†
Information and Software Engineering
George Mason University
Fairfax VA 22030-4400 USA
(+1) 703-993-1654
www.ise.gmu.edu/~ofut/
ofut@ise.gmu.edu



*Thirteenth International Conference on Engineering of Complex Computer Software, Annapolis, MD, November 2002.*


## Abstract


*A commonly held belief is that applications written in object-oriented languages are more modular than those written in procedural languages. This paper presents results from an experiment that examines this hypothesis. Open source and industrial program modules written in the procedural languages of Fortran and C were compared with open source program modules written in the object-oriented languages of C++ and Java. The metrics examined in this study were lines of code per module and number of parameters per module. The results of the investigation support the hypothesis. The modules of the object-oriented programs were found to be half the size of those of the procedural programs and the average number of parameters per module for the object-oriented programs was approximately half that of the procedural programs. Thus the object-oriented programs were twice as modular as the procedural programs. An unexpected result was that the C++ programs were found to be no more modular than the C programs.*


## 1. Introduction

Most current software development projects are using some sort of object-oriented design and an object-oriented programming language. *Object-oriented programming* is based on the concept of an object, which is a data structure encapsulated with a set of routines, called methods, which operate on the data [1, 8]. Object-oriented programming evolved from *structured programming*, which emphasizes top-down development and modular structure, so that each module performs a specific function [5]. Object-oriented programming was intended, among other things, to be more modular and help programmers develop modules that exhibit less coupling and more cohesion. It is widely believed that applications that are more modular are also more maintainable, extendable and reusable [1, 4, 5, 6]. This paper tries to establish a relation between modularity and object-oriented programming for industrial software modules.

This paper presents results from a study designed to compare the modularity of programs written in procedural languages to those written in object-oriented languages. The hypothesis investigated is that object-oriented programs are more **modular** than procedural programs. Exactly what the property of modularity means is somewhat open to discussion. One would like to say that modularity is a property of the entire software application, but that is a difficult characterization to quantify. One informal characterization is that modularity is related to the ease of analyzing or modifying a piece of code in isolation from the rest of the application. This, of course, is related to the common metrics of coupling and cohesion [2]. Although methods to calculate these metrics have been discussed [7], no method is widely agreed on.

Thus, this study examined three attributes associated with modularity. For the purposes of this experiment, the term *module* refers to a subroutine or function in Fortran and a method in C, C++ and Java (sometimes also called "unit"). The attributes studied are the size of the modules, the number of modules, and the number of parameters passed to a module. A program with a few large modules is considered to be *less* modular than a program with many, small modules. In addition, a large number of parameters may indicate that a module is too complex and should be subdivided [5].

The remainder of this paper presents the design of our experiment, results from our experiments, analysis of the results, and some limitations of our results. In conclusion, we found that object-oriented programs did indeed exhibit


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

more modularity. However, a surprising result is that C++ programs do **not** exhibit more modularity than C programs. We theorize that this indicates that (1) the benefits of object-oriented programming are derived primarily from the design rather than the use of a particular language, and (2) many C++ programmers are not following a true object-oriented design.

## 2. Experimental Design

This experiment analyzed 38 applications. Ten applications were written in Fortran, ten in C, ten in C++ and eight in Java. Fortran and C are procedural languages, while C++ and Java are object-oriented languages. Most applications were obtained as open source code arbitrarily selected from those available on the Internet, however finding open source Fortran applications was difficult and only three were found. The first author had access to several Fortran applications that had been developed in-house and seven of these were included to bring the count to ten. The appendix lists the programs used and where they were obtained. The applications that were investigated are of a variety of different types. Combining the results from a number of different application types for each language should provide results that can be extrapolated to the general case. However, it would also be interesting to compare applications of the same type. Since web server applications were available that had been written in C, C++ and Java, an additional comparison was made between the three of them.

Several tools were used to analyze the programs selected. The author created a tool to extract the number of lines of code per module and the number of parameters per module for Fortran programs. *Understand for C++* (Scientific Toolworks, Inc., www.scitools.com) was used to count the number of lines of code per method in C and C++ programs. Scientific Toolworks also provided a perl script to extract the number of parameters per method from the database files that their tool created. *JStyle* (Man Machine Systems, www.mmsindia.com) was used to determine the number of lines of code per method in Java programs. JStyle provides an integrated scripting capability that the author used to extract the number of parameters passed per method.

This experiment investigated two hypotheses:

**1**. Object-oriented programs have more, smaller modules than procedural programs. Since the programs investigated are not all the same size, it was decided to use number of modules per thousand lines of code as a measure of the number of modules. The size of modules was calculated as the number of lines of code per module.

**2**. The average number of parameters per module is smaller for object-oriented programs than for procedural programs.

## 3. Results

The tools described previously were used to measure the number of modules, number of lines of code per module and number of parameters per module. Descriptive statistics were calculated for the number of lines of code per module and the number of parameters per module.

### 3.1 Experiment 1: Module Size Comparison

The number of lines of code per module was determined for 38 applications, 10 in each of Fortran, C and C++, and 8 in Java. The number of modules and lines of code (LOC) per module were determined, as well as the minimum, maximum, mean, median, mode and standard deviation of the lines of code per module [3, 8, 9]. In addition, the number of modules per thousand lines of code (KLOC) was also calculated. This metric was used to normalize the measure of number of modules per application for comparison between applications of different sizes. The tabulated results for the applications written in Fortran, C, C++, and Java are displayed in Tables 1 through 4.

### 3.2 Experiment 2: Number of Parameters

As noted in the Introduction, a module is defined to be a function, procedure, or method. The number of parameters per module was determined for the same 38 applications. The minimum, maximum, mean, median, mode and standard deviation of the number of parameters were calculated. The tabulated results for applications written in Fortran, C, C++, and Java are displayed in Tables 5 through 8, respectively.

## 4. Analysis

Fenton and Pfleeger [3] point out that the mean, median and mode of the data will be the same in a normal distribution. An examination of the collected data clearly shows that they do not form normal distributions. Therefore, normal parametric statistical analyses such as linear regression and the t-test cannot be applied.

**Table 1. Module size comparison for applications written in Fortran**

| | | | LOC/ module | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Application | # Modules | Total LOC | Mean | Min | Max | Median | Mode | Std Dev | Mod/KLOC |
| advsrc | 22 | 829 | 37.7 | 3 | 274 | 13 | 4 | 65.0 | 26.5 |
| anypia | 216 | 5453 | 25.3 | 1 | 436 | 12 | 6 | 49.1 | 39.6 |
| booster | 26 | 398 | 15.3 | 3 | 122 | 9.5 | 10 | 23.8 | 65.3 |
| fdwell | 16 | 626 | 39.1 | 9 | 62 | 41 | 50 | 15.9 | 25.6 |
| image | 35 | 1434 | 41.0 | 3 | 146 | 19 | 6 | 45.1 | 24.4 |
| lowtran | 93 | 5585 | 60.1 | 2 | 718 | 21 | 12 | 99.1 | 16.7 |
| mlsc | 115 | 7831 | 68.1 | 2 | 812 | 32 | 9 | 100.5 | 14.7 |
| plume | 43 | 901 | 21.0 | 4 | 62 | 18 | 4 | 18.2 | 47.7 |
| quail | 240 | 9816 | 40.9 | 1 | 264 | 27 | 11 | 42.3 | 24.4 |
| simfence | 563 | 28199 | 50.1 | 1 | 3489 | 25 | 4 | 156.3 | 20.0 |
| All applications | 1369 | 61072 | 44.6 | 1 | 3489 | 21 | 4 | 112.0 | 22.4 |

**Table 2. Module size comparison for applications written in C**

| | | | LOC/ module | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Application | # Modules | Total LOC | Mean | Min | Max | Median | Mode | Std Dev | Mod/KLOC |
| apache | 1689 | 49239 | 29.2 | 0 | 636 | 15 | 4 | 44.3 | 34.3 |
| chemtool | 234 | 9993 | 42.7 | 4 | 983 | 17 | 8 | 93.9 | 23.4 |
| gcompris | 365 | 9861 | 27.0 | 0 | 451 | 13 | 9 | 40.7 | 37.0 |
| gdpc | 76 | 2864 | 37.7 | 3 | 587 | 7 | 4 | 87.6 | 26.5 |
| gperiodic | 120 | 3124 | 26.0 | 0 | 404 | 10 | 7 | 46.7 | 38.4 |
| gstat | 1276 | 34904 | 27.4 | 2 | 961 | 16 | 5 | 49.2 | 36.6 |
| gsx | 87 | 3308 | 38.0 | 4 | 503 | 16 | 4 | 65.0 | 26.3 |
| mutt | 1214 | 48124 | 39.6 | 0 | 1267 | 19 | 8 | 74.5 | 25.2 |
| umfpack | 97 | 12518 | 129.1 | 0 | 828 | 80 | 13 | 160.4 | 7.7 |
| xpaint | 865 | 22138 | 25.6 | 1 | 253 | 15 | 5 | 31.4 | 39.1 |
| All applications | 6023 | 196073 | 32.6 | 0 | 1267 | 16 | 7 | 59.6 | 30.7 |

**Table 3. Module size comparison for applications written in C++**

| | | | LOC/ module | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Application | # Modules | Total LOC | Mean | Min | Max | Median | Mode | Std Dev | Mod/KLOC |
| amaya | 5641 | 258718 | 45.9 | 0 | 1937 | 24 | 4 | 75.3 | 21.8 |
| benson | 70 | 1100 | 15.7 | 0 | 103 | 10.5 | 4 | 19.3 | 63.6 |
| btl2 | 389 | 4487 | 11.5 | 0 | 88 | 8 | 2 | 12.9 | 86.7 |
| celestia | 1014 | 14472 | 14.3 | 0 | 364 | 5 | 4 | 28.1 | 70.1 |
| gperf | 100 | 2920 | 29.2 | 1 | 339 | 12.5 | 1 | 52.2 | 34.2 |
| guarddog | 89 | 1597 | 17.9 | 0 | 228 | 7 | 3 | 32.6 | 55.7 |
| knetfilter | 139 | 5007 | 36.0 | 2 | 212 | 17 | 2 | 44.3 | 27.8 |
| ktouch | 126 | 1766 | 13.8 | 1 | 142 | 5 | 4 | 22.1 | 71.3 |
| pi3web | 2113 | 41023 | 19.4 | 0 | 680 | 8 | 0 | 41.9 | 51.5 |
| skysight | 2954 | 89175 | 30.2 | 0 | 6914 | 11 | 4 | 229.9 | 33.1 |
| All applications | 12635 | 420265 | 33.4 | 0 | 6914 | 14 | 4 | 124.3 | 30.1 |

**Table 4. Module size comparison for applications written in Java**

| | | | | LOC/ module | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Application | # Modules | Total LOC | | Mean | Min | Max | Median | Mode | Std Dev | Mod/KLOC |
| ArtOfIllusion | 2993 | 41022 | | 13.7 | 0 | 602 | 3 | 1 | 31.5 | 73.0 |
| bugbase | 1143 | 7511 | | 6.6 | 0 | 223 | 1 | 1 | 14.3 | 152.2 |
| ConsultComm | 158 | 1485 | | 9.4 | 0 | 205 | 2 | 1 | 21.2 | 106.4 |
| jakarta-tomcat | 5669 | 41433 | | 7.3 | 0 | 312 | 1 | 1 | 17.0 | 136.8 |
| jCharts | 370 | 2120 | | 5.7 | 0 | 72 | 1 | 1 | 10.4 | 174.5 |
| jext | 3296 | 33155 | | 10.1 | 0 | 906 | 2 | 1 | 33.2 | 99.4 |
| jigsaw | 7427 | 55710 | | 7.5 | 0 | 308 | 2 | 1 | 13.8 | 133.3 |
| mercator | 3428 | 11926 | | 3.5 | 0 | 126 | 1 | 1 | 7.7 | 287.4 |
| All applications | 24484 | 194362 | | 7.9 | 0 | 906 | 2 | 1 | 20.6 | 126.0 |

**Table 5. Number of parameters per module in Fortran programs**

| | | Parameters/ Module | | | | | |
|---|---|---|---|---|---|---|---|
| Application | # Modules | Mean | Min | Max | Median | Mode | Std Dev |
| booster | 26 | 3.0 | 1 | 7 | 3 | 2 | 1.5 |
| fdwell | 16 | 6.0 | 2 | 19 | 5 | 4 | 4.2 |
| image | 35 | 7.4 | 1 | 18 | 7 | 6 | 4.7 |
| lowtran | 93 | 4.1 | 0 | 17 | 3 | 2 | 3.1 |
| mlsc | 115 | 2.9 | 0 | 34 | 2 | 2 | 4.1 |
| plume | 43 | 4.9 | 1 | 18 | 4 | 1 | 4.2 |
| simfence | 563 | 4.1 | 0 | 18 | 4 | 3 | 2.7 |
| advsrc | 22 | 1.7 | 0 | 7 | 1 | 1 | 1.6 |
| quail | 240 | 3.5 | 0 | 27 | 3 | 0 | 3.9 |
| anypia | 216 | 1.4 | 0 | 8 | 1 | 0 | 1.7 |
| All applications | 1369 | 3.6 | 0 | 34 | 3 | 3 | 3.3 |

**Table 6. Number of parameters per module in C programs**

| | | Parameters/ Module | | | | | |
|---|---|---|---|---|---|---|---|
| Application | # Modules | Mean | Min | Max | Median | Mode | Std Dev |
| apache | 1689 | 2.3 | 0 | 10 | 2 | 2 | 1.4 |
| chemtool | 234 | 2.2 | 0 | 11 | 2 | 2 | 1.9 |
| gcompris | 365 | 1.3 | 0 | 14 | 1 | 1 | 1.5 |
| gperiodic | 120 | 1.5 | 0 | 14 | 1 | 1 | 1.7 |
| gstat | 1276 | 2.2 | 0 | 11 | 2 | 2 | 1.7 |
| gsx | 87 | 1.8 | 0 | 5 | 2 | 2 | 2.0 |
| mutt | 1214 | 2.5 | 0 | 14 | 2 | 1 | 1.8 |
| xpaint | 865 | 2.5 | 0 | 10 | 2 | 3 | 1.5 |
| gdpc | 76 | 2.2 | 0 | 17 | 2 | 2 | 1.9 |
| umfpack | 97 | 4.6 | 0 | 16 | 3 | 2 | 3.7 |
| All applications | 6023 | 2.3 | 0 | 17 | 2 | 1 | 1.7 |

**Table 7. Number of parameters per module in C++ programs**

| | | Parameters/ Module | | | | | |
|---|---|---|---|---|---|---|---|
| Application | # Modules | Mean | Min | Max | Median | Mode | Std Dev |
| btl2 | 389 | 1.3 | 0 | 15 | 1 | 1 | 1.7 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| celestia | 1014 | | 1.3 | 0 | 40 | 1 | 0 | 2.0 |
| gperf | 100 | | 1.1 | 0 | 6 | 1 | 0 | 1.4 |
| guarddog | 89 | | 0.9 | 0 | 4 | 1 | 1 | 0.9 |
| knetfilter | 139 | | 0.4 | 0 | 2 | 0 | 0 | 0.7 |
| ktouch | 126 | | 0.8 | 0 | 6 | 0 | 0 | 1.3 |
| amaya | 5641 | | 2.6 | 0 | 21 | 2 | 1 | 2.2 |
| benson | 70 | | 1.4 | 0 | 6 | 1 | 1 | 1.2 |
| pi3web | 2113 | | 1.6 | 0 | 12 | 1 | 0 | 1.6 |
| skysight | 2954 | | 2.3 | 0 | 14 | 2 | 1 | 1.9 |
| All applications | 12635 | | 2.2 | 0 | 40 | 2 | 1 | 2.0 |

**Table 8. Number of parameters per module in Java programs**

| | | | Parameters/ Module | | | | | |
|---|---|---|---|---|---|---|---|---|
| Application | # Modules | | Mean | Min | Max | Median | Mode | Std Dev |
| ArtOfIllusion | 2993 | | 1.4 | 0 | 25 | 1 | 1 | 1.9 |
| bugbase | 1143 | | 0.9 | 0 | 13 | 1 | 0 | 1.1 |
| ConsultComm | 158 | | 0.7 | 0 | 4 | 1 | 1 | 0.8 |
| jakarta-tomcat | 5669 | | 0.9 | 0 | 11 | 1 | 0 | 1.1 |
| jCharts | 370 | | 1.0 | 0 | 7 | 1 | 0 | 1.4 |
| jext | 3296 | | 0.8 | 0 | 7 | 1 | 0 | 1.0 |
| jigsaw | 7427 | | 1.0 | 0 | 7 | 1 | 0 | 1.1 |
| mercator | 3428 | | 0.5 | 0 | 7 | 0 | 0 | 0.8 |
| All applications | 6023 | | 0.9 | 0 | 25 | 1 | 0 | 1.2 |

### 4.1 Analysis of Module Size Comparison

An examination of the data in Tables 1 through 4 shows noticeable variation among the individual applications and between those written in the different programming languages. It is likely that there is a significant variation in lines of code per module across the entire population of programs. Combining the results from the object-oriented applications and from the procedural applications may provide results that are representative of the general case. Table 9 summarizes the aggregated data, while Figure 1 provides a graphical comparison of the module sizes for the object-oriented, procedural, Java, C++, C, and Fortran applications.

A major purpose of this experiment is to investigate whether object-oriented programs have "more" modules that are smaller than those in similarly sized procedural programs. Comparing the size of the modules is straightforward. The average size of the object-oriented modules (16.6 lines of code) is slightly less than half (47%) the size of the procedural modules (34.8 lines of code). Comparing the number of modules cannot be done directly because of the differences in sizes of the applications. To normalize the data, the number of modules per thousand lines of code is instead used as the metric for comparison. The number of modules per thousand lines of code for the object-oriented applications is just over twice (210%) the number for the procedural applications. These results support the idea that object-oriented applications are more modular than procedural applications.

**Table 9. Summary of module size comparison**

| | | | | LOC/ Module | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Language | # Modules | Total LOC | | Mean | Min | Max | Median | Mode | Std Dev | | Mod/KLOC |
| Procedural | 7392 | 257145 | | 34.8 | 0 | 3489 | 17 | 4 | 72.4 | | 28.7 |
| Object-oriented | 37121 | 614627 | | 16.6 | 0 | 6914 | 5 | 1 | 75.4 | | 60.4 |

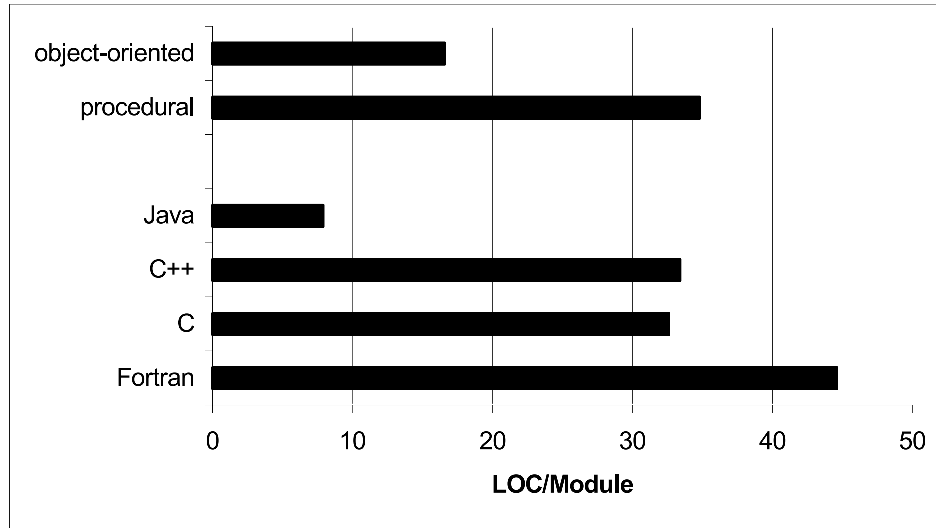


**Figure 1. Graphical comparison of module size**

Since three of the applications examined are web servers written in three different languages, it might be instructive to compare their results. As they were all written for the same purpose, they might be more similar to each other than to the other programs examined. Table 10 compares the results for these three applications. As can be seen, there is more difference between the two object-oriented applications (C++ and Java) than there is between the C application and the C++ application. It is not obvious therefore that comparing similar applications is any more useful than comparing dissimilar applications.

An examination of the overall results for the four languages, given in Table 11, shows that the average size of the C++ applications investigated is very close to that of the C applications examined (33.4 LOC/module vs. 32.6 LOC/module). Correspondingly, the modules per thousand lines of code, which is used as a normalized measure of the number of modules per application, is also quite similar (30.1 vs. 30.7). This suggests that the C++ programs that were examined were no more modular than the C programs. Two complementary theories may explain this. Although C++ supports object-oriented programming, it does not strongly encourage the programmer to use an object-oriented approach, as does Java. Additionally, many C++ programmers are re-trained C programmers who have had little or no exposure to OO concepts and thus cannot be expected to use the OO language features effectively. These two related observations may indicate that, these C++ applications may not have been written using an object-oriented approach.

**Table 10. Comparison of three web-server applications**

| | | | | LOC/ Module | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Language | Application | # Modules | Total LOC | Mean | Min | Max | Median | Mode | Std Dev | Mod/KLOC |
| C | apache | 1689 | 49239 | 29.2 | 0 | 636 | 15 | 4 | 44.3 | 34.3 |
| C++ | pi3web | 2113 | 41023 | 19.4 | 0 | 680 | 8 | 0 | 41.9 | 51.5 |
| Java | jigsaw | 7427 | 55710 | 7.5 | 0 | 308 | 2 | 1 | 13.8 | 133.3 |

**Table 11. Module size summary**

| | | | LOC/Module | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Language | # Modules | Total LOC | Mean | Min | Max | Median | Mode | Std Dev | Mod/KLOC |
| Fortran | 1369 | 61072 | 44.6 | 1 | 3489 | 21 | 4 | 112.0 | 22.4 |
| C | 6023 | 196073 | 32.6 | 0 | 1267 | 16 | 7 | 59.6 | 30.7 |
| C++ | 12635 | 420265 | 33.4 | 0 | 6914 | 14 | 4 | 124.3 | 30.1 |
| Java | 24484 | 194362 | 7.9 | 0 | 906 | 2 | 1 | 20.6 | 126.0 |

## 4.2 Analysis of Number of Parameters

Examination of Tables 5-8 indicates a significant variability in the number of parameters per module. Table 12 summarizes the data for the applications written in the four languages, and also shows the aggregated summary for the procedural and object-oriented applications. Not surprisingly, the minimum number of parameters per module is zero, regardless of the programming language. The mean number of parameters ranges from a low of 0.9 for Java programs to a high of 3.3 for Fortran programs. As in Experiment 1, the results for the C and C++ programs are quite close. The C programs had an average of 2.3 parameters per module, while the C++ programs had an average of 2.2. For the applications analyzed in this experiment there seems to be little difference in modularity between the C and C++ programs.

Comparison of the aggregated results for the procedural and object-oriented applications shows that there is a noticeable difference. The maximum number of parameters was 40 for the object-oriented programs and 34 for the procedural programs. However, the standard deviations of the measurements indicate that there was more variability in number of parameters for the procedural programs than for the object-oriented programs. For the object-oriented programs the mean number of parameters was 1.3, but the mode was 0. For the procedural programs, the mean was 2.5 with a mode of 1. Thus, the procedural programs averaged approximately twice as many parameters per module as the object-oriented programs. These results are compared graphically in Figure 2.

Another way to look at the results is to examine the distribution of the number of parameters. That is, we decided to count the number, or percent, of modules that have a given number of parameters. The percentage data is presented in Figure 3 for the Fortran, C, C++, and Java programs, for up to ten parameters. Figure 3 is read as follows. About 43% of the Java modules had zero parameters, and about 36 had one parameter. It appears that the Fortran programs have the largest number of parameters per module and the Java programs the smallest. However, the distribution of parameters is very different for the applications written in the four different languages, making it difficult to directly compare the results.

Another way to look at the results is to examine the distribution of the number of parameters. That is, we decided to count the number, or percent, of modules that

**Table 12. Parameters/module summary**

| | | Parameters/ Module | | | | | |
|---|---|---|---|---|---|---|---|
| Language | # Modules | Mean | Min | Max | Median | Mode | Std Dev |
| Fortran | 1369 | 3.6 | 0 | 34 | 3 | 3 | 3.3 |
| C | 6023 | 2.3 | 0 | 17 | 2 | 1 | 1.7 |
| C++ | 12635 | 2.2 | 0 | 40 | 2 | 1 | 2.0 |
| Java | 24484 | 0.9 | 0 | 25 | 1 | 0 | 1.2 |
| procedural | 7392 | 2.5 | 0 | 34 | 2 | 1 | 2.1 |
| object-oriented | 37121 | 1.3 | 0 | 40 | 1 | 0 | 1.6 |

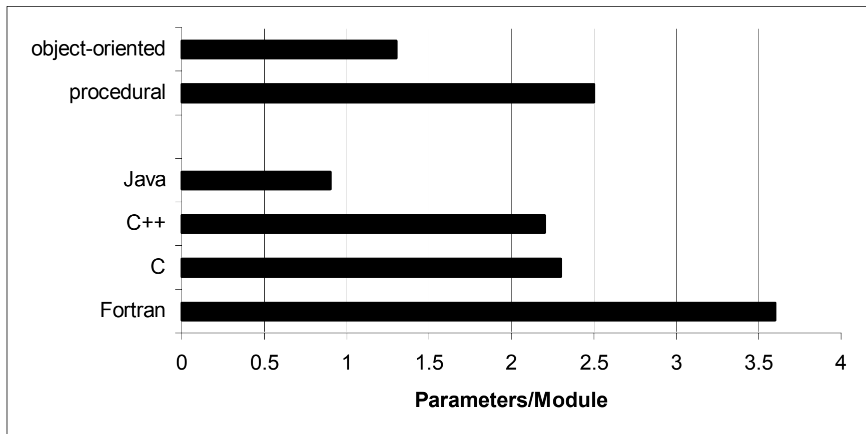


**Figure 2. Graphical comparison of parameters per module**

have a given number of parameters. The percentage data is presented in Figure 3 for the Fortran, C, C++, and Java programs, for up to ten parameters. Figure 3 is read as follows. About 43% of the Java modules had zero parameters, and about 36 had one parameter. It appears that the Fortran programs have the largest number of parameters per module and the Java programs the smallest. However, the distribution of parameters is very different for the applications written in the four different languages, making it difficult to directly compare the results.

Another approach is to look at cumulative percentages, or percentiles. This information is presented in Figure 4. Approximately 28% of the Fortran modules, 35% of the C modules, 46% of the C++ modules, and 81% of the Java modules have at least one parameter. Perhaps a more useful way to compare these numbers is to look at them from the inverse, that is, to examine the percentage of modules that have more than one parameter. For the Fortran programs this is 72%, for the C programs it is 65%, for the C++ programs it is 54%, and for the Java programs it is 19%. This is a striking difference. There is a nearly four-fold difference between Fortran and Java.

Figure 5 compares the distribution of parameters for the aggregated data for the procedural and object-oriented modules. It is clear from this figure that the object-oriented modules have fewer parameters.

Another approach is to look at cumulative percentages, or percentiles. This information is presented in Figure 4. Approximately 28% of the Fortran modules, 35% of the C modules, 46% of the C++ modules, and 81% of the Java modules have at least one parameter. Perhaps a more useful way to compare these numbers is to look at them from the inverse, that is, to examine the percentage of modules that have more than one parameter. For the Fortran programs this is 72%, for the C programs it is 65%, for the C++ programs it is 54%, and for the Java programs it is 19%. This is a striking difference. There is a nearly four-fold difference between Fortran and Java.

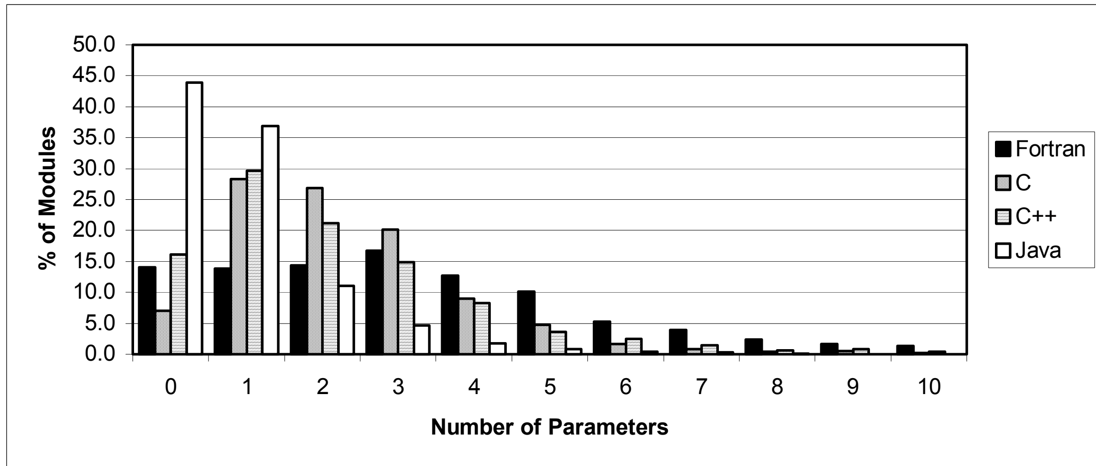


**Figure 3. Distribution of number of parameters**

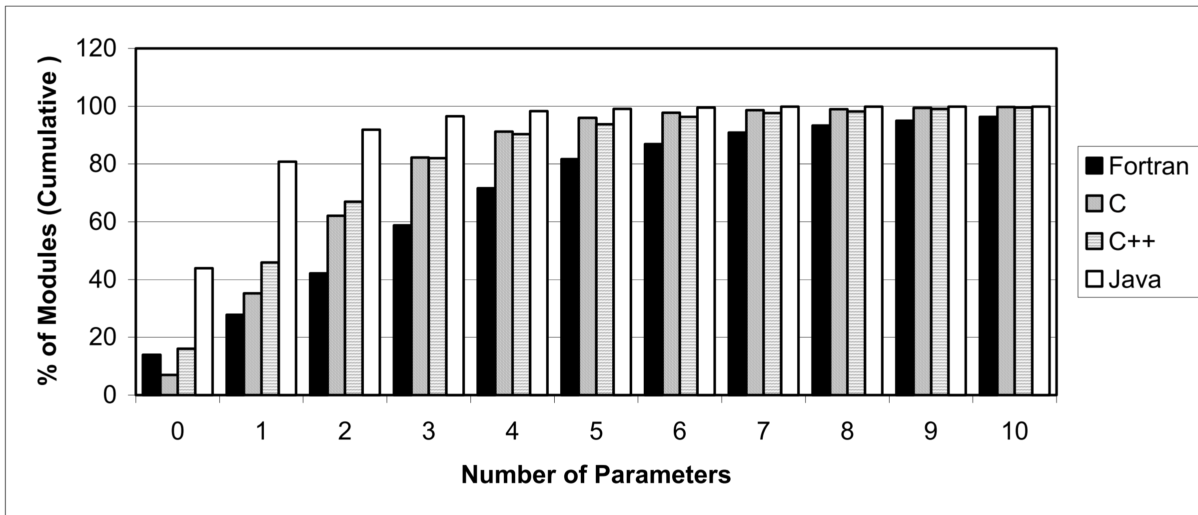


**Figure 4. Percentiles for number of parameters**

Figure 5 compares the distribution of parameters for the aggregated data for the procedural and object-oriented modules. It is clear from this figure that the object-oriented modules have fewer parameters.

Figure 6 shows the cumulative percentages, or percentiles, for the aggregated data for the procedural and object-oriented modules. Approximately 35.5% of the procedural modules have at most one parameter while approximately 70% of the object-oriented modules have at most one parameter. This means that approximately 64.5% of the procedural modules have more than one parameter and only about 30% of the object-oriented modules have more than one parameter. By this measure, the object-oriented modules were more than twice as modular as the procedural modules. However, if the results for the object-oriented programs were skewed to the low side as a result of being open source code, as discussed previously for Experiment 1, the difference for mature programs might not be as large as that measured here.

## 5. Assumptions and Limitations

There are several limitations inherent in this study. A perennial problem with software engineering experimentation is whether the subject programs are representative of the entire population. The applications we used were arbitrarily chosen from those available and may or may not be representative.

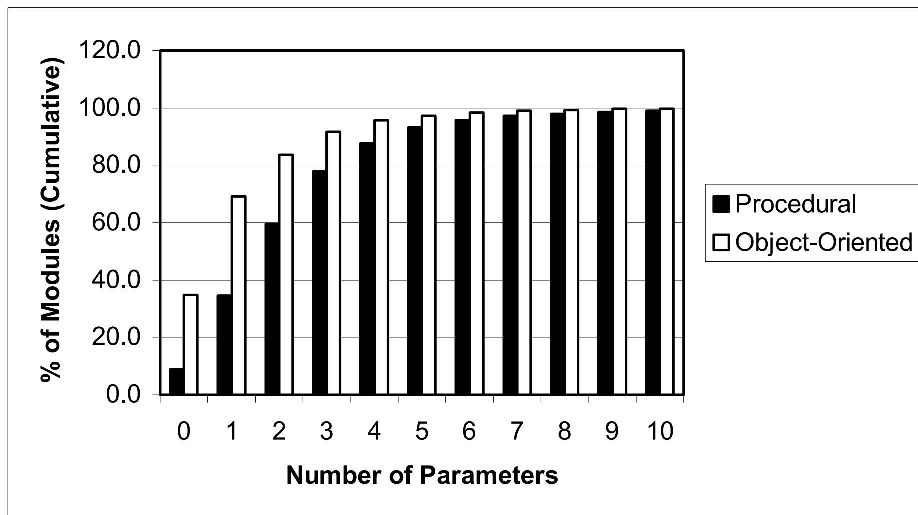


**Figure 5. Distribution of number of parameters (aggregated data)**

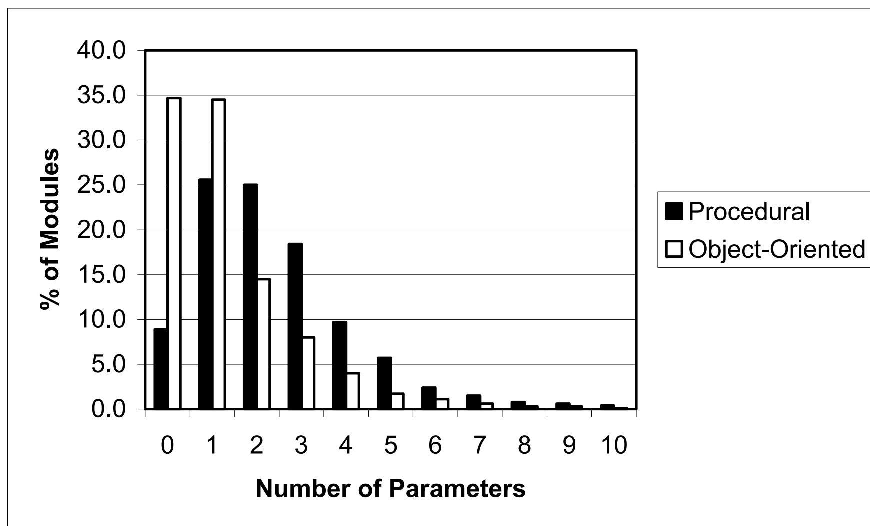


**Figure 6. Percentiles for number of parameters (aggregated data)**

Most of the programs used were open source applications. It was observed, though not discussed previously in this paper, that some of the applications that were analyzed had fifty or more modules with no lines of code. It may be that these applications are still in development and may not be representative of more mature code. If there are a significant number of incomplete modules in these programs, it is possible that the average lines of code per module for mature Java applications, and possibly C++ applications, may be larger than that measured in this study. Correspondingly, the number of parameters for mature applications written in these languages may also be understated. It could be instructive to repeat this experiment with a different set of applications that might be more mature.

In addition, the applications that were chosen almost certainly represent both command line and GUI-based applications. A majority of the Fortran programs in this study are command line applications, whereas many of the other programs are almost certainly GUI-based. This difference may have an impact on the results. Further studies could investigate whether there is a difference in modularity between these types of applications.

Another observation is that the ages of the applications are not all the same. In particular, the Fortran programs were written in the 1980s, a time when the field of software engineering was just beginning to understand the concept of modularity. Programs written more recently in Fortran might well be more modular. Unfortunately, we had none available.

Another concern is that just because a program is written in C++ does not necessarily mean that it was designed using an object-oriented approach. Some C programs are "ported" to C++ with the intent of making them more object-oriented as work progresses. The results obtained do suggest that the C++ programs examined are substantially less modular than the Java programs. In fact, there seems to be little difference in modularity between the C++ programs and the C programs. Future work might be done with a different language. SmallTalk, Modula2, Ada, and Object Pascal are designed to enforce object-oriented programming [8] and may be better choices for a comparison between applications written in object-oriented and procedural languages.

Fortran, C, C++ and Java programs were chosen for this experiment because they are widely used and thus of interest to a large segment of the IT industry. Investigation of programs written in other languages would provide valuable additional information.

## 6. Conclusions

The results of the study are consistent with, and provide support for both of the experimental hypotheses. The first hypothesis, that object-oriented programs have more, smaller modules than procedural programs, was supported in the first experiment. The object-oriented programs had twice as many modules as the procedural programs, when normalized on the basis of modules per thousand lines of code. The average module size of the object-oriented programs, measured as lines of code, was half that of the procedural programs. The second hypothesis, that object-oriented programs have, on average, fewer parameters per module than procedural programs, was supported in the second experiment. The procedural applications had approximately twice as many parameters/module, on average, as the object-oriented applications. The results therefore strongly support the idea that object-oriented programs are more modular than procedural programs. In this experiment, the object-oriented programs were approximately twice as modular as the procedural programs.

An unexpected and potentially significant result was that, in this study at least, the C++ applications were no more modular than the C applications. This suggests that the programmers who created the C++ applications that were investigated in this study did not, in fact, use an object-oriented approach.

## References:


[1] G. Booch, *Object-Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[2] L. L. Constantine and E. Yourdon, *Structured Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[3] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, 2nd Edition, PWS Publishing Company, Boston, Massachusetts, 1997.

[4] B. Henderson-Sellers, *A Book of Object-Oriented Knowledge*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1992.

[5] J. K. Hughes and J. I. Michtom, *A Structured Approach to Programming*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[6] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1994.

[7] A. J. Offutt, M. J. Harrold and P. Kolte, A Software Metric System for Module Coupling", The Journal of Systems and Software, March 1993, 20(3):295-308.

[8] R. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object Oriented Modeling and Design*, Prentice Hall, 1991.

[9] M. Sheppard, *Foundations of Software Measurement*, Prentice Hall International, Hemel, Hempstead, Hertfordshire, England, 1995.