8

Changing Programming Paradigm - An Empirical Investigation

James Miller, Gerald Darroch, Murray Wood, Andrew Brooks and Marc Roper

Dept. Computer Science, University of Strathclyde,
Livingstone Tower, Richmond Street, Glasgow G1 1XH, Scotland

Abstract

This paper outlines an investigation into the ability of experienced programmers to effectively change programming paradigm. In particular it investigates programmers changing from imperative to object-oriented programming. This is achieved by comparing code samples from the new and experienced object-oriented programmers. Each sample is presented to a tool which makes a number of automatic measurements. The comparison has revealed large style differences between the two sets of programmers. The two groups having very different ideas on what constitutes an object, resulting in significant variations in the concepts of the responsibility or extent of a class, and inheritance and associated operations. These differences will typically result in major variations in the quality and reusability of the software produced.

Key Codes: D1.5; D2.8
Keywords: Object-Oriented Programming; Metrics

1. INTRODUCTION

Object-oriented techniques are viewed by many as the most exciting development within the field of software construction. Companies and organisations are rushing to retrain their technical personnel in this new medium and in the process change experienced imperative programmers into novice object-oriented programmers. But what are the consequences? Can experienced proficient imperative programmers be considered ready to construct new systems after a short re-orientation or does the change in paradigm require time to mature with the retrained personnel? Programmers are expected to master additional concepts (such as inheritance, function overloading and polymorphism) and successfully integrate them into their existing 'toolkit' of techniques for system construction. Will the retrained personnel produce 'better' systems or will it lead to a reduction in quality? This study attempts to help answer these questions.

2. OVERVIEW OF THE EXPERIMENT

The main objective behind this project is an attempt to characterise the principal differences (in terms of coding style) between novice and experienced object-oriented programmers. It should be noted that the novice object-oriented programmers are themselves experienced programmers in other paradigms). In order to achieve this goal the project team carried out an empirical evaluation on code produced by both novice and experienced subjects. Evaluation took the form of submitting code samples from both groups to an attributes tool(OOATS) (produced for this project). The tool calculates various measures[1, 2, 3, 4] for each sample of work and produces a 'characterisation' of the code in terms of various object-oriented properties. The properties can be grouped into three overlapping categories: responsibility or extent, complexity and inheritance properties of an object. These properties were subsequently

M. Lee et al. (eds.), *Software Quality and Productivity*

summed for each group, and the two summations were statistically compared for significant differences.

The selection of a set of measures to characterise an object-oriented program is (currently) a subjective process. Little is understood about what contributes to good or bad object-oriented code to fully characterise the process. Hence we have chosen a list of measures as a starting point. It is hoped as the study is extended the number and type of measurements will increase. No claim is made that this set of measurements provide any particular insight, simply an interesting basis for exploration.

3. EXPERIMENTAL DESIGN AND ANALYSIS

3.1 Subjects and their Experience

The study has collected sample code from both camps: novice and experienced. Novice object-oriented programmers were people who had at least three and as many eight years traditional programming expertise, but had only recently transferred paradigm. All programmers in the novice group were experienced C programmers. The code selected was their first attempt at producing a significant (> 1000 lines) object-oriented system. In addition it was verified that the programmers' objective was to produce 'good quality' object-oriented code and not just a working solution. Programmers in the experienced object-oriented group all had three years or more object-oriented experience. All programmers in the experienced group were experienced C++ programmers. The study was carried out in C++. It was hoped to minimise any ability and familiarisation factors by the choice of languages and participants' backgrounds.

3.2 Measurements

All measurements were automatically produced via an object-oriented metrics tool produced for this study. The tool measured the following characteristics for each program:

Average number of methods per class
Average number of instance variables per class
Average size of method per class
Average class 'complexity'
Average method 'complexity'
Average instances of function overloading per class
Average instances of operator overloading per class
Average number of 'children' classes
Maximum depth of inheritance per program

The method 'complexity' is calculated using McCabe's Cyclomatic number (as suggested by Graham[3]) and the class complexity is the sum of the class' method complexities as suggested by Chidamber and Kemerer[1]. All other measurements are self-explanatory.

3.3. Analysis

The results from a t-test on each measurement will be described in this section. The t-test was selected because of its effective performance with small samples[5]. From the collected programs, 31 samples from each camp were identified for analysis, via random sampling. The Null hypothesis is "that the means of the two samples are equal" (i.e. We have found no difference between the novice and experienced programmers) and hence the alternative hypothesis is "that the means are different" (i.e. we have found a difference). The t value for each measurement is:

Table 1. t-test values for each measurement

Measurement	*t Value*
Average number of methods per class	5.069*
Average number of instance variables per class	8.307*
Average size of method per class	0.312
Average class 'complexity'	5.250*
Average method 'complexity'	0.485
Average instances of function overloading per class	3.715*
Average instances of operator overloading per class	5.355*
Average number of 'children' per class	5.700*
Maximum depth of inheritance per program	5.970*

* means the null hypothesis has been rejected and the alternative accepted. All results are stated at the standard 5% significance level.

4. DISCUSSION AND CONCLUSIONS

Although great care must be taken to draw general conclusions from experiments with small sample sizes, we believe the results show that significant differences (in style) exist between novice and experienced object-oriented programmers. In seven out of the nine attributes measured our experiment has found a statistical difference. The 2 attributes where no difference was detected (Average size of method per class, Average method 'complexity') are attributes with directly comparative attributes within imperative programming (Average function size, Average function 'complexity'). Hence one might surmise that total programming experience rather than specific experience within an individual paradigm is the most important factor and hence due to the two camps having similar experience no difference would be expected.

These results are contrary to the study by Wilde and Huitt[6], they reported finding (in professional commercial work) median method sizes of "1 executable statement for C++ programs and 3 noncommented lines for Smalltalk programs". Although not explicitly measured by the tool, the average size of methods in our study is approximately 10 uncommented statements (in both novice and experienced). This is closer to typical figures for imperative systems. (Estimates for imperative systems vary greatly depending on experience of programmers, language used, etc. Typically, they are even larger, e.g. in a study by Basili and Perricone[7], they quote an average of approximately 100 lines of commented Fortran).

The significant differences in the other measurements seem to indicate that the novice and experienced subjects have very different ideas on what constitutes an object. The novice programmers are producing a series of relatively independent objects with functionality exactly meeting the specification. Whereas the experienced programmers are producing larger objects (with additional functionality) fully integrated into their class hierarchies. For example, the novice programmers produced objects with 6 methods on average (range 1 to 23) whereas the experienced programmers produced objects with 25 methods on average (range 1 to 131); the novice programmers produced objects with, on average, less than one instance of function overloading (range 0 to 5) whereas the experienced programmers produced objects with, on average, three instances of function overloading (range 0 to 16). See Table 2 for a full description of averages and ranges.

We could surmise that the experienced programmers are overspecifying their system in an attempt to provide reusable components (additional methods, operator and function overloading - resulting in increased class 'complexity'). This in turn allows them to easily build an extensive hierarchy (with their reusable components) and in turn accounts for their increased use of inheritance (number of children, maximum depth of inheritance).

Table 2. Averages and ranges for each measurement

Measurement	*Novice*		*Experienced*	
	Ave.	Range	Ave.	Range
Average number of methods per class	6	23	25	131
Average number of instance variables per class	4	16	2	11
Average size of method per class	10	37	9	35
Average class 'complexity'	13	77	38	245
Average method 'complexity'	1.5	5	1.5	14
Average instances of function overloading per class	<1	5	3	16
Average instances of operator overloading per class	<1	1	4	17
Average number of 'children' per class	0.1	3	0.5	8
Maximum depth of inheritance per program	0.2	2	1	4

In conclusion, this study casts doubt that organisations using novice programmers will in the short term reap the perceived benefits of object-oriented construction and suggests that the paradigm change is not a straightforward process. It further suggests the programmers will require a period of time to adapt their style and thinking patterns to accommodate the new paradigm. We would again warn against generalising the results of this study due to its limited sample size. Currently we are investigating extending the study both in terms of sample size and measurements collected, such as connascence[8] and the Law of Demeter[9].

REFERENCES

1. CHIDAMBER, S.R. AND KEMERER, C.F. Towards a metrics suite for Object-Oriented design. In *OOPSLA*, 1991, pp 197-211.
2. COPPICK, J. AND CHEATHAM, J. *Software Metrics for Obejct-Oriented Systems.* ACM press, 1992.
3. GRAHAM, I. *Object-Oriented Methods.* Addison-Wesley, 1993.
4. LORENZ, M. *Object-Oriented software development: a practical guide,* Prentice-Hall, 1993.
5. BULMER, M.B. *Principles of statistics.* Dover, 1979.
6. WILDE, N. AND HUITT, R. Maintenance support for object-oriented programs. *IEEE Transactions on Software Engineering,* 1992, Vol.18, (12), pp 1038-1044.
7. BASILI, V.R. AND PERRICONE, B.T. Software errors and complexity. In M. Shepperd, editor, *Software Engineering Metrics Volume 1: Measures and Validations,*.1993, pp 168-183, McGraw-Hill.
8. PAGE-JONES, M. Comparing techniques by means of encapsulation and connascence. *Communications of the ACM,* 1992, vol.35, pp 147-152.
9. LIEBERHERR, K.L. AND HOLLAND, I.M. Assuring good style for object-oriented programs. *IEEE Software,* 1989, vol 6, pp 38-48.