Short Papers

An Empirical Investigation of the Influence of a Type of Side Effects on Program Comprehension

J.J. Dolado, *Member*, *IEEE*,
M. Harman, *Member*, *IEEE*,
M.C. Otero, and L. Hu

Abstract—This paper reports the results of a study on the impact of a type of side effect (SE) upon program comprehension. We applied a crossover design on different tests involving fragments of C code that include increment and decrement operators. Each test had an SE version and a side-effect-free (SEF) counterpart. The variables measured in the treatments were the number of correct answers and the time spent in answering. The results show that the side-effect operators considered significantly reduce performance in comprehension-related tasks, providing empirical justification for the belief that side effects are harmful.

Index Terms—Side-effect-free programs, crossover designs, program comprehension.

1 Introduction

A side effect is any change in program state that occurs as a by-product of the evaluation of an expression. Side effects are often thought to impede program comprehension, although hitherto, this appears to be a belief that has not been examined empirically. This paper is concerned with a type of side effect in C programs and their effect upon program comprehension.

The C programming language standard defines side effects in a very broad manner ([9, Section 5.1.2.3]): "Accessing a volatile object, modifying an object, modifying a file, or calling a function that does any of those operations are all side effects, which are changes in the state of the execution environment. Evaluation of an expression may produce side effects." In this paper, we adopt a more restricted view, considering only auto-increment and auto-decrement operators. A side effect will be taken to be: "Any change in the value of a variable which occurs when an expression is evaluated, other than an assignment expression-statement."

A side-effect-free program is one which contains no side effects. This defines side effects to occur in expressions like "`x++`," "`- -v`," "`(x+1,y=x+3)`," and "`x==2 && y=2*x`." Assignment statements in C are assignment expression-statements. To be side-effect free, assignment expressions are allowed to change only the value of the variable on the lefthand side of the assignment; the righthand side must not change the value of any variables.

Our basic research enquiry is centered around the question: "Are side-effect-free programs more understandable than programs with side-effects?" This paper uses a side effect removal algorithm, LinSERT [6], to produce side-effect-free versions of C program fragments which contain side effects. The tool is used to avoid the potential source of bias inherent in human selection of side-effect-free programs. LinSERT's algorithm was designed to remove side effects, not to produce readable code. Indeed, there are cases when, subjectively, one could argue that the results obtained by LinSERT are not ideally readable. However, this observation only serves to strengthen the results obtained.

1.1 Related Work

Side effects are widely believed to inhibit program understanding, with a consequent detrimental impact upon software maintenance and evolution. For example, Kernighan and Pike [10] suggest that side effects should only be used in very special situations, where well-understood side effect idioms are employed to improve a program's performance. Other authors, for example, [1], advise caution, suggesting that the programmer should carefully consider "the tradeoff between increased speed and decreased maintainability that results when embedded assignments are used in artificial places."

The study of side effects is related to studies concerning the effects on program comprehension of the syntactic presentation of the program code, which in turn affects readability. Miara et al. [15] showed that indentation has a significant effect on comprehension for both experienced and novice programmers. Oman and Cook [19] studied the effect on program comprehension of source code formatting and commenting. Related to the issue of studying a language construct, we may be in a similar situation to other assertions about potential dangers of some practices. The first of the "considered harmful" saga began with the seminal work of Dijkstra [4], which spun a new and fresh approach to programming, as well as a decade of fruitful discussions. The type of side effects that we are dealing with here can be also labeled as one in the list of activities suggested "harmful." In some cases, the practices have the form of a "taboo" [14]. The activity of reading code plays a basic role in the development and maintenance activities; therefore, any improvement in the procedures, methods, languages, and tools for increasing the comprehensibility of the code will have direct effects on the progress of the software building activities [7].

In the rest of the paper, we present the details of the experiment and its analysis. In Section 2, we describe the experiment, the design, hypotheses, and other elements. Section 3 presents the analysis of the data. Finally, Section 4 states the conclusions.

2 Formal Experiment

2.1 Crossover Designs for the Experiment

We considered two types of programs, SEF versus SE programs, and we planned a parallel study between two groups. In order to have the two groups getting both treatments (SEF and SE), we performed a crossover design in which the two groups received both treatments but in different order. Applying just one treatment for each subject would have the drawback that the variation of the measurements between subjects could distort the true effect of the treatment. By taking measures of each individual in both treatments, we avoid this variability, although we may incur other problems, such as detecting other effects apart from the treatment.

Crossover designs, in which each subject receives a sequence of treatments (repeated measures), are a well-known type of experimental design, used in clinical and medical studies [5], [8], [12]. The most widely used design of this type is the two treatment, two-period (two-round) crossover design. This structure represents, for instance, first, the administration of two drugs to the two groups and then, after withdrawal, the reverse application of the

- *J.J. Dolado and M.C. Otero are with the Department of Computer Languages and Systems, University of the Basque Country, 20.009-San Sebastián, Spain. E-mail: jipdocoj@si.ehu.es, jipotvim@vc.ehu.es.*
- *M. Harman and L. Hu are with the Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK. E-mail: {mark.harman, Lin.Hu}@brunel.ac.uk.*

Manuscript received 28 May 2002; revised 12 Mar. 2003; accepted 17 Mar. 2003.
Recommended for acceptance by G. Canfora.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 116634.

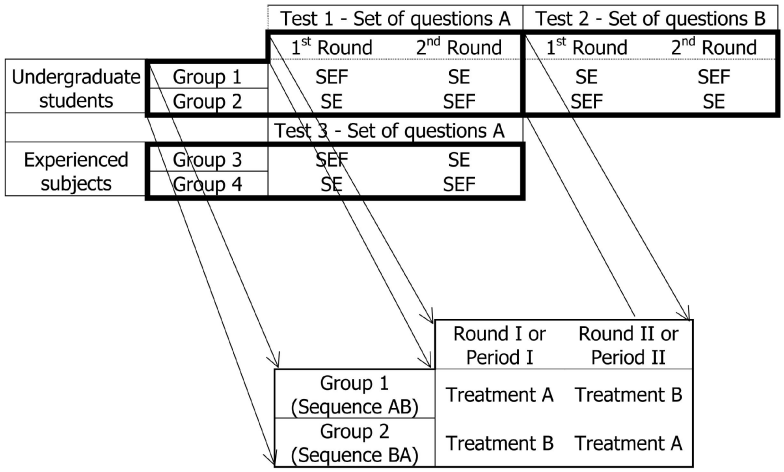

Fig. 1. Layout of the complete experiment. Each of the three bold-edged zones is a test formulated as a crossover design.

treatments. However, these well-known designs present several problems, both in their use of terminology and in their analysis of data [3].

2.2 Factors and Dependent Variables

We will examine the existence of the usual effects of a crossover design: the treatment effects (TREAT), the sequence or order of application of the treatments (GROUP), the effects of the period (or round) of application (PERIOD), and the residual effect of the first period (or round) into the second (CARRYOVER). The group or sequence effect, GROUP, measures the residual difference between the two sequences of application: 1) SEF then SE; 2) SE then SEF. The period effect, PERIOD, measures the existence of differences due to the two rounds of treatment (due to learning, maturation, or other causes). The paramenter treatment, TREAT, is the direct effect of applying SEF or SE. The carryover effect, CARRYOVER, is the residual effect of applying the treatment after the first period. This effect has to be included in the equations modeling the second period. In clinical studies, it represents the residual effects of the treatment of the first period which persist into the second period, contaminating the effects of the second treatment. For the sake of clarity, PERIOD effects are not considered as CARRYOVER effects, although, in several texts, both effects are named as residual or carryover effects, indistinctly.

We designed two separate sets of questions (set of questions A for Tests 1 and 3, set of questions B for Test 2) with a different level of difficulty. The dependent variables that are measured as response to the questions are the number of correct answers (SCORE) and the time spent in answering them (TIME).

Fig. 1 depicts the crossover designs used in the three days. In each crossover design, the effects due to the treatment variable TREAT (SEF versus SE) are not confounded: neither with the effects of GROUP nor with the PERIOD effect (maturation or learning effects). Therefore, we can measure the GROUP effects (order of presentation) and the learning effects. However, as mentioned above, there is confusion with some interactions. The crossover design for the second day is similar to that of the first day, but the order of the tests was reversed in each group and used the set of questions B.

2.3 Experimental Procedure

The material prepared, the two different sets of questions having each one the two aspects SEF-SE, was delivered on two different days. On the first day, Test 1 (with the set of questions A) was carried out, and Test 2 (with the set of questions B) was completed on the second day (one week later). On each day, the sequence of treatments was presented to the two different groups of students and the experimental design corresponds to a crossover design of repeated measures in the dependent variables. There are two options in the treatment, SEF or SE, implying two sequences of treatments and two periods of treatment (each day). In the repetition of the experiment with experienced subjects (Test 3), only the material corresponding to the set of questions A was delivered.

The obvious problem with this design (in the current setting) is that, by taking the two versions of the experimental material, using the same subjects on the same day consecutively, we could incur the risk that, in the second period, some effects of the first period could remain. Either after the first version of the test, the subjects could perform better because they are more acquainted with the process or they could perform worse due to fatigue or other nuisance factors. We have not investigated in detail the psychological processes that could be involved in this situation. Instead, we analyzed the possibility of the existence of design effects, as suggested by the experimental design literature [3]. The expected duration of the tests was low (less than one hour) and the existence of those effects should not be overestimated.

2.4 Materials and Threats to the Experiment

The materials provided were an initial exam for classifying the ability of the subjects and the two actual set of questions A (simple C fragments) and B (more complex programs) (see Fig. 2 for some samples and the companion Web page for the full documents), with two versions each (SEF and SE), delivered in the two days, with students of the third and fourth year of the BS degree in computer science as experimental subjects. The subjects were split into two groups. A second set of experienced professionals took only the first set of questions A. The level of expertise in the experienced group was mixed; it was split into two subgroups with similar characteristics, using self-assessment.

The threats to the experiment were carefully considered by creating groups with equal skills, by measuring the possible learning effects (PERIOD), and by taking into account the effects of the CARRYOVER from the first period into the second. The issues of history, plagiarism, and generalization of the results were also considered in the design.

2.5 Hypotheses

The statistical hypotheses derived from the general question stated in the introduction which are explored here are:

1. Null hypotheses for the treatments in the tests: There is no significant difference between the means of the SCOREs

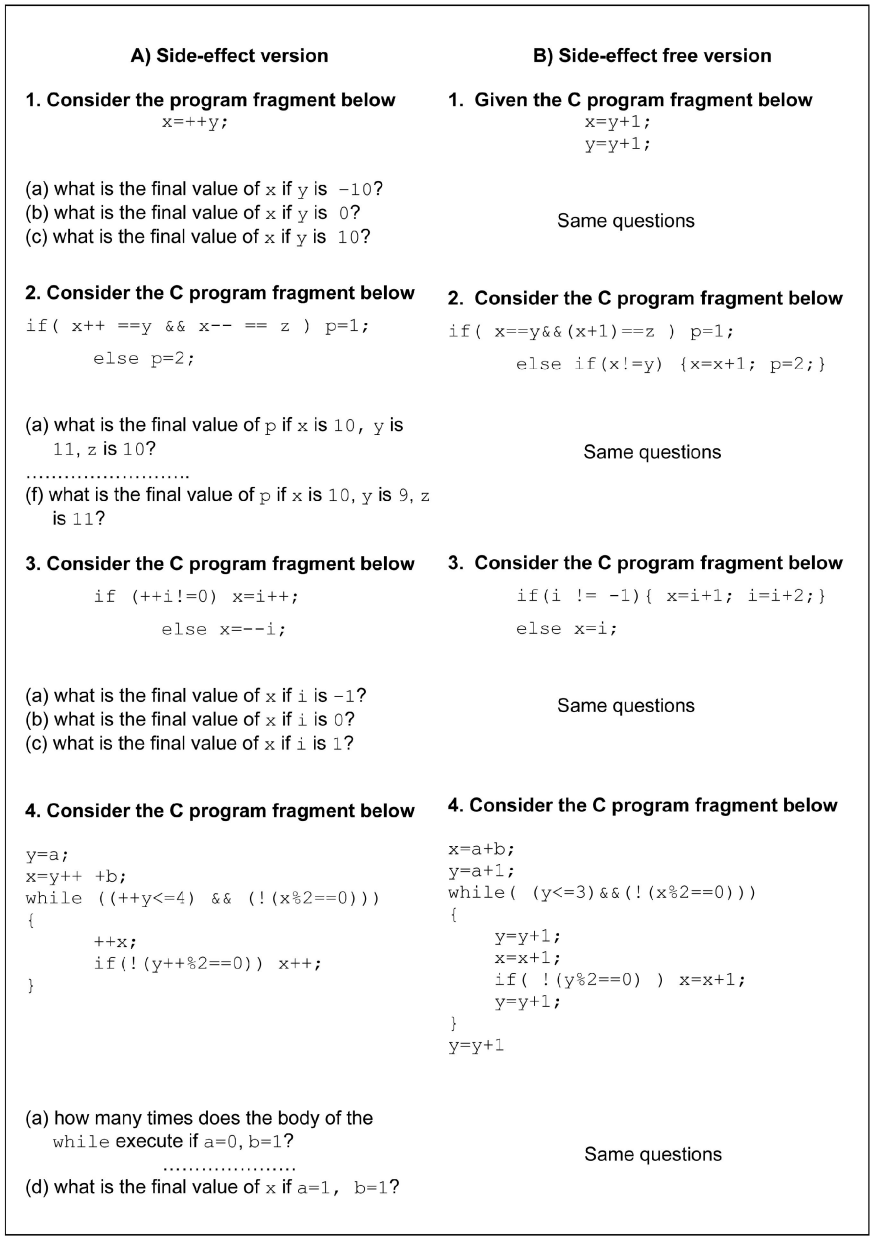

Fig. 2. Sample questions in the tests (both versions).

obtained in Test 1, using the set of questions A, with respect to the effects of the two levels of the treatment: SEF and SE versions ($H_{0-T1-SCO-TREAT}$). The same null hypothesis is stated for the TIMEs ($H_{0-T1-TIM-TREAT}$). Similarly, for Test 2, we test the null hypotheses $H_{0-T2-SCO-TREAT}$ and $H_{0-T2-TIM-TREAT}$.

2. Null hypotheses for the periods and for the groups: We state that there is no significant difference between the means of the SCOREs obtained in the tests with respect to the two different PERIODs of treatment ($H_{0-T1-SCO-PERIOD}$) and ($H_{0-T2-SCO-PERIOD}$) and with respect to the two GROUPs (equivalent to testing the CARRYOVER effects, $H_{0-T1-SCO-GROUP}$, and $H_{0-T2-SCO-GROUP}$). The same null hypotheses are stated for the TIMEs in both tests ($H_{0-T1-TIM-PERIOD}$, $H_{0-T2-TIM-PERIOD}$, $H_{0-T1-TIM-GROUP}$, and $H_{0-T2-TIM-GROUP}$).

Test 3 has the corresponding set of hypotheses.

2.6 Power Analysis and Sample Size

As prescribed in experimental design, the identification of the sample size, the effect size, and the desired power is a requirement before conducting the experiment [11], [13], [17]. We performed an "a priori" power analysis, exploring the relationships among the sample size (n), the effect size (f), the significance level (α), and the desired power ($1-\beta$) since we could not change, for instance, the sample size n.

We have to remember that α is the probability of committing a type I error. That is, to reject, incorrectly, the null hypothesis when it is true. As the process of transforming SE into SEF is performed automatically, the cost that we would incur when committing a type I error can be safely downsized (no human cost is involved in the transformation process). The type I error means to assume that SEF and SE programs have different effects when they actually have not. When automation is unavailable, the cost involved is the additional effort, if there is any, of side-effect-free coding. On the other hand, committing a type II error, that is, to accept,

TABLE 1
ANOVA Results for Test 1, Test 2, and Test 3

ANOVA results for Test 1					
Source	Measure	F	Sig.	η_p^2	Obs. power
Within subjects					
TREAT	SCORE	181.226	0.000	0.919	1.000
	TIME	25.029	0.000	0.610	0.999
PERIOD	SCORE	0.581	0.457	0.035	0.189
	TIME	8.096	0.012	0.336	0.859
Between subjects					
GROUP	SCORE	0.729	0.406	0.044	0.211
	TIME	0.819	0.379	0.049	0.224
ANOVA results for Test 2					
Source	Measure	F	Sig.	η_p^2	Obs. power
Within subjects					
TREAT	SCORE	34.892	0.000	0.714	1.000
	TIME	7.131	0.018	0.337	0.814
PERIOD	SCORE	0.431	0.522	0.030	0.165
	TIME	6.155	0.026	0.305	0.762
Between subjects					
GROUP	SCORE	0.716	0.412	0.049	0.208
	TIME	1.431	0.251	0.093	0.309
ANOVA results for Test 3					
Source	Measure	F	Sig.	η_p^2	Obs. power
TREAT	SCORE	37.245	0.000	0.741	1.000
	TIME	14.070	0.002	0.520	0.971
PERIOD	SCORE	0.267	0.614	0.020	0.141
	TIME	7.163	0.019	0.355	0.813
Between subjects					
GROUP	SCORE	0.367	0.555	0.027	0.155
	TIME	1.498	0.243	0.103	0.317

incorrectly, the null hypothesis when it is false, would imply that we will be wasting human effort as we would be using the SE versions instead of taking advantage of their SEF counterparts. The commission of a type II error, represented by β, means to assume that there are not differences between SEF and SE when in fact there are.

In the "a priori" power analysis, we followed the approach of Stevens ([20]). One of our constraints was the absence of previous data about the effect size and the second one was that our sample size was fixed (no more than 18 subjects); therefore, we explored the combinations of n, Cohen's f [2], α (set at 0.1), and $1-\beta$ (set at 0.8) that could render the experiment valid. A more precise analysis was carried out in the "a posteriori" analysis, once we had all the actual data, by following the approach of Yue and Roach [22], who provide a formula for computing n. The final results provide values for n ranging from 1 to 3, with extreme values around 12, which are below the number of subjects we used. Therefore, the results of the tests can be statistically trusted, according to both approaches to power analysis.

3 Statistical Analysis

By performing an analysis of repeated measures in the factors TREAT (direct effect), PERIOD and GROUP ([18, pp. 435-440]), we obtained the results summarized in Table 1, using the F statistic (computed with $\alpha = 0.1$), which has high F-values in the SCORE variables of the treatments. Other variables that are presented are the significance of the F-test (Sig.), the effect size, η_p^2 (computed in SPSS), and the observed power of the test. Fig. 3 shows the comparative results obtained in the three experimental tests.

The null hypotheses for Test 1 that are not rejected are $H_{0-T1-SCO-GROUP}$ (F = 0.729, p = 0.406), $H_{0-T1-SCO-PERIOD}$, and $H_{0-T1-TIM-GROUP}$ (see Table 1). In the variable TIME, there are learning effects due to PERIOD ($H_{0-T1-TIM-PERIOD}$ is rejected with F = 8.096, p = 0.012), implying that, once the experiment began, some kind of "learning" or ability in the subjects was increased and affected the time measured in the second period, given the syntactical differences of the versions. The treatment has significant results for SCORE and TIME (both $H_{0-T1-SCO-TREAT}$ and $H_{0-T1-TIM-TREAT}$ are rejected). Therefore, the conclusion is that the use of SEF has a clear impact on the SCORE.

For Test 2 (see Figs. 3c and 3d), the situation remains similar to Test 1: There is a statistically significant difference between the two treatments SEF and SE with respect to both SCORE and TIME. The treatment is acting directly on SCORE ($H_{0-T2-SCO-TREAT}$ is rejected with F = 34.892, p = 0.000) as there are no residual effects detected. There are residual effects in PERIOD ($H_{0-T2-TIM-PERIOD}$ is rejected) with respect to the TIME variable.

The set of questions A were repeated with more experienced people in Test 3 (see Figs. 3e and 3f). This group did not receive any previous training about the experimental procedures. With the values reported in Table 1, we can make the following inferences:

1. There are neither GROUP (or CARRYOVER) effects in SCORE nor in TIME.
2. There are no PERIOD effects in SCORE (F = 0.267, p = 0.614), but there are PERIOD effects in the response variable TIME (F = 7.163, p = 0.019).
3. There are clear effects of the treatment on the variable SCORE and the null hypothesis of no effects is rejected. The null hypothesis for TIME is also rejected, but there are learning effects.

The conclusion is that the behavior of the experienced subjects is similar to that of the more inexperienced and the SEF code was more understandable than the corresponding SE counterparts.

3.1 Discussion

We have rejected the null hypothesis for the nonexistence of effects of the treatment SEF versus SE. In the analysis, we have examined the existence of the different residual effects of the crossover designs. We did not find any carryover effects of the crossover. The boxplots of Fig. 3 graphically show the differences in the treatment effects in the three settings, in both SCORE and TIME.

However, an unexpected effect that was found was that there is a recurrent period effect in the TIME measure in the crossover design of each day and in the repetition. In clinical or drug studies, similar effects are justified by the action of the drug under test. In our case, it is unknown what the mechanism is (possibly psychological or some kind of "concentration" in the task) that causes the times spent in the second period to be affected by the activities of the first one. The study of this question lies beyond the scope of this article.

4 Conclusions and Future Work

The purpose of this research was to study the potential benefits of using side-effect-free programs in place of side-effecting counterparts. We designed two different sets of questions which were presented to the experimental subjects in two versions. The first one contained elementary questions about SE programs. We have observed more correct answers and in less time in the SEF version. The set of questions B had greater complexity than the set of questions A, but we have observed similar results. The repetition of Test 1 corroborated the initial conclusions. The transformation of the SE code into SEF code was performed automatically by means of the LinSERT tool, but there remains the question, to be explored in the future, of what the differences would be if the SE programs had been transformed manually.

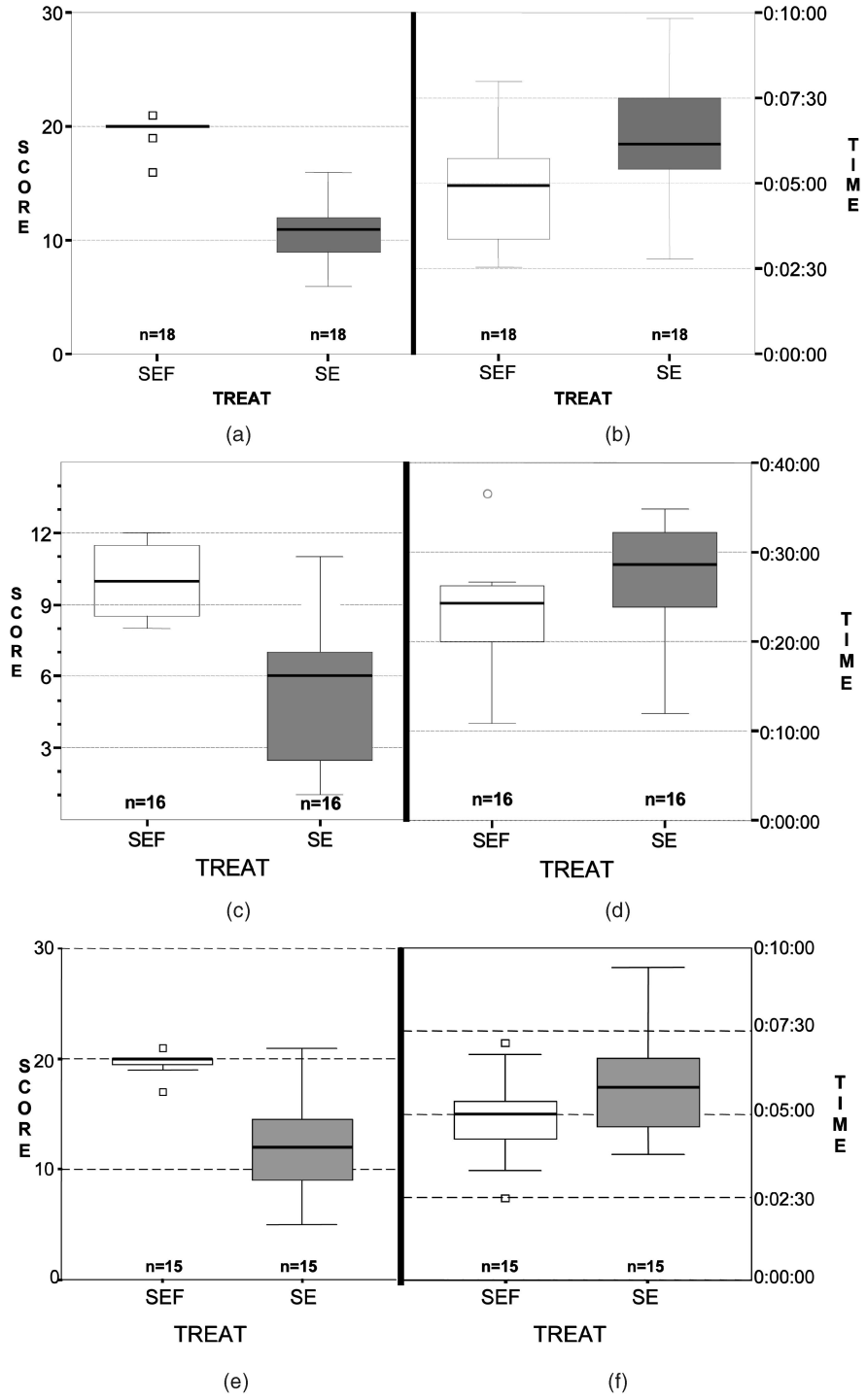

Fig. 3. (a) and (b) Boxplots for Test 1. (c) and (d) Boxplots for Test 2. (e) and (f) Boxplots for Test 3.

This work does not try to add a new taboo to the software engineering community [14], but to add sound conclusions, based on solid empirical practice [21], to the list of heuristics which tend to reduce the complexity of software [16]. It is noteworthy to see how such a simple program fragment can have strong effects on comprehension. The results show that there is a clear and significant degradation of performance for these limited (intraprocedural) forms of side effect. More complex forms of side effects, such as interprocedural side effects, may produce similar (or worse) degradation in performance, but this remains a topic for future investigation. A side lesson is that the constructs, or building blocks, used when defining a new programming language should be tested. From the experimental viewpoint, the cost involved in assessing the use of a new language by a community of programmers is low compared to the benefits obtained by avoiding the risky effects of some programming practices.

ACKNOWLEDGMENTS

This work has been supported, as appropriate, by EPSRC grants GR/M58719, GR/58719, GR/R98938, GR/M78083, by Daimler

Chrysler AG, Berlin, by CICYT TIC 1143-C03-01, UPV-EHU EA-8099 and Diputación Gipuzkoa SP-Kudea. The authors would like to thank Martin Shepperd, their students, and the members of the SEMINAL network, the programming group of the University de Sevilla, and the anonymous referees for their invaluable comments. The experimental material and the data used in this article can be found at http://www.sc.ehu.es/jiwdocoj/sef/sef.htm.

REFERENCES

[1] L. Cannon, R. Elliott, L. Kirchhoff, J. Miller, J. Milner, R. Mitze, R. Schan, E. Whittington, N. Spencer, H. Keppel, D. Brader, and M. Brader, "Recommended C Style and Coding Standards," http://www.cs.umd.edu/users/cml/cstyle/indhill-cstyle.html, 2000.
[2] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences,* second ed. Lawrence Erlbaum Assoc., 1988.
[3] G.E. Dallal, "The Computer-Aided Analysis of Crossover Studies," http://www.tufts.edu/~gdallal/crossovr.htm, 2000.
[4] E.W. Dijkstra, "Goto Statement Considered Harmful," *Comm. ACM,* vol. 11, no. 3, pp. 147-148, Mar. 1968.
[5] J.E. Grizzle, "The Two-Period Change-Over Design and Its Use in Clinical Trials," *Biometrics,* vol. 21, pp. 467-480, 1965.
[6] M. Harman, L. Hu, X. Zhang, and M. Munro, "Side-Effect Removal Transformation," *Proc. IEEE Int'l Workshop Program Comprehension (IWPC 2001),* pp. 309-319, May 2001, available at http://www.brunel.ac.uk/~csstmmh2/linsert/.
[7] D. Hendrix, J.H. Cross II, and S. Maghsoodloo, "The Efectiveness of Control Structure Diagrams in Source Code Comprehension Activities," *IEEE Trans. Software Eng.,* vol. 28, no. 5, pp. 463-477, May 2002.
[8] M. Hills and P. Armitage, "The Two-Period Cross-Over Clinical Trial," *British J. Clinical Pharmacology,* vol. 8, pp. 7-20, 1979.
[9] ISO, International Standards Organisation: Programming Languages—C, International standard, ISO/IEC 9899: 1990 (E), Dec. 1990.
[10] B.W. Kernighan and R. Pike, *The Practice of Programming.* Addison-Wesley, 1999.
[11] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El-Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Trans. Software Eng.,* vol. 28, no. 8, pp. 721-734, Aug. 2002.
[12] R.O. Kuehl, *Design of Experiments: Statistical Principles of Research Design and Analysis,* second ed. Duxbury Thomson Learning, 2000.
[13] R.V. Lenth, "Some Practical Guidelines for Effective Sample Size Determination," *The Am. Statistician,* vol. 55, no. 3, pp. 187-193, 2001.
[14] L.F. Marshall and J. Webber, "Gotos Considered Harmful and other Programmers' Taboos," Technical Report 699, Dept. of Computing Science, Univ. of Newcastle, 2000.
[15] R.J. Miara, J.A. Musselman, J.A. Navarro, and B. Shneiderman, "Program Indentation and Comprehensibility," *Comm. ACM,* vol. 1983, no. 11, pp. 861-867, Nov. 1983.
[16] S. McConnell, "Keep It Simple," *IEEE Software,* vol. 13, no. 6, pp. 127-128, Nov. 1996.
[17] J. Miller, J. Daly, M. Wood, M. Roper, and A. Brooks, "Statistical Power and Its Subcomponents—Missing and Misunderstood Concepts in Empirical Software Engineering Research," *Information and Software Technology,* vol. 39, no. 4, pp. 285-295, 1997.
[18] G.A. Milliken and D.E. Johnson, *Analysis of Messy Data, volume 1: Designed Experiments.* Cambridge Univ. Press, 1984.
[19] P.W. Oman and C.R. Cook, "Typographic Style Is More than Cosmetic," *Comm. ACM,* vol. 33, no. 5, pp. 506-520, May 1990.
[20] J. Stevens, *Applied Multivariate Statistics for the Social Sciences,* third ed. Lawrence Erlbaum Assoc., 1996.
[21] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* Kluwer Academic, 2000.
[22] L.Q. Yue and P. Roach, "A Note on the Sample Size Determination in Two-Period Repeated Measurements Crossover Design with Application to Clinical Trials," *J. Biopharmaceutical Statistics,* vol. 8, no. 4, pp. 577-584, 1998.

Comments on "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics"

William M. Evanco

Abstract—It has been proposed that size should be taken into account as a confounding variable when validating object-oriented metrics. We take issue with this perspective since the ability to measure size does not temporally precede the ability to measure many of the object-oriented metrics that have been proposed. Hence, the condition that a confounding variable must occur causally prior to another explanatory variable is not met. In addition, when specifying multivariate models of defects that incorporate object-oriented metrics, entering size as an explanatory variable may result in misspecifed models that lack internal consistency. Examples are given where this misspecification occurs.

Index Terms—Object-oriented metrics, software defects, defect-proneness, statistical modeling.

1 INTRODUCTION

IN [1], El Emam et al. raise issues regarding the validation of object-oriented metrics. The claim is made that to properly validate these metrics, we must statistically control for size since size is often correlated with the metrics. We wish to dispute this claim.

Software size, for example as measured by source lines of code, is one of the earliest measures of software characteristics. It is a metric that is easily collected during compile time and is well defined for most programming languages [12]. As such, some analysts regard it as sacrosanct and a benchmark against which all other metrics must be compared. The issues that El Emam et al. have raised with regard to object-oriented metrics are not new, having been posed in the past for metrics characterizing procedural languages. For example, many of the software characteristic measures, such as cyclomatic complexity and Halstead volume, have been shown to be highly correlated with source lines of code [6].

In a similar vein, El Emam et al. examine the Chidamber and Kemerer object-oriented metrics [3] as well as a subset of the Lorenz and Kidd metrics [11] on the basis of a large C++ telecommunications system. They demonstrate, that many of the OO metrics having a significant relationship to defect-proneness (see [1, Fig. 3]) also have a significant association with source lines of code as measured by the Spearman correlation coefficient (see [1, Table 4]). A C++ class is regarded as defect-prone if it exhibits one or more defects during the operational phase. They point out that "studies to date have relied exclusively on univariate analysis to test the hypothesis that the product metrics are associated with fault-proneness or the number of faults." According to them, if software size is regarded as a confounding effect, then this measure must enter into any statistical analysis looking at the impact of some object-oriented metric on fault-proneness. After size is entered as a confounding variable, if the object-oriented metric has a parameter value that is statistically significant, then, in their view, the metric is a "valid" object-oriented measure of defect-proneness.

• *The author is with the College of Information Science and Technology, Drexel University, 3141 Chestnut St., Philadelphia, PA 19104. E-mail: william.evanco@cis.drexel.edu.*

Manuscript received 14 Jan. 2002; revised 6 Aug. 2002; accepted 17 Feb. 2003. Recommended for acceptance by N. Schneidewind.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 115693.