Comments on "Formal Methods Application: An Empirical Tale of Software Development"

Daniel M. Berry and Walter F. Tichy

Abstract—We comment on the experimental design and the result of the paper mentioned in the title. Our purpose is to show interested readers examples of what can go wrong with experiments in software research and how to avoid the attending problems.

1 Introduction

Empirical studies and controlled experiments in particular have become an important tool for understanding the nature and efficacy of software methods and tools. A positive trend in recent years has been that the number of papers with empirical data published in *IEEE Transactions on Software Engineering* (*TSE*) and elsewhere has been increasing. This trend is motivated in part by the realization that, unlike in the early days of software research, mere demonstration of a new tool or method is not enough. There is a bewildering variety of software engineering methods and the relative merits of competing approaches are poorly understood. Furthermore, the methods and their interactions with the real world of software development are too complicated to be understood by theory alone. Actual observation of programmers in realistic settings is beginning to go hand in hand with the development of new methods and techniques, thus putting software research on a firm footing.

In this vein, it is heartening to see the experiment by Sobel and Clarkson [5]. The experiment collected evidence that "formal methods students had increased complex problem solving skills" and that "the use of formal analysis during software development produces 'better' programs." Formal methods have a long history of theoretical research, but rigorous, empirical evaluation is scarce. Pfleeger and Hatton published a case study [3] on formal methods with inconclusive results; their paper points to additional case studies in this area. Sobel and Clarkson report on the first quasi experiment on formal methods.

Unfortunately, the paper contains several subtle problems. The reader unfamiliar with the basic principles of experimental psychology may easily miss them and interpret the results incorrectly. Not only do we wish to point out these problems, but we also aim to illustrate what to look for when drawing conclusions from controlled experiments. We thus hope to help both experimenters and readers of empirical software research to become more astute in regards to meaningful experimentation in software research.

Much has been written about experimental methodology; a classic text is the book by Christensen [2]. The book covers a wide range of experimental principles, including control, experimental design, data collection, validity, ethics, and hypothesis testing. However, since the book is written for psychologists, it may appear dry and inaccessible to software researchers and practitioners. However, by using the experiment by Sobel and Clarkson as a concrete example, many of these principles come into sharp focus. Never have we read Christensen with more interest than in the context of the Sobel and Clarkson experiment! We hope that this note will help motivate computer scientists to study with renewed interest the body of knowledge about experimental design.

- *D.M. Berry is with the School of Computer Science, University of Waterloo, 200 University Ave., West Waterloo, Ontario N2L 3G1, Canada. E-mail: dberry@haifa.math.uwaterloo.ca.*
- *W.F. Tichy is with the Department of Informatics, University of Karlsruhe, 76128 Karlsruhe, Germany. E-mail: tichy@ira.uka.de.*

Manuscript received 12 July 2002; accepted 20 Feb. 2003.
Recommended for acceptance by D. Rosenblum.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 116936.

2 Overview of Published Paper

In the following discussion, the published paper [5] by Sobel and Clarkson is referred to as "the *TSE* paper." The personal pronoun "we" refers to the authors of the present note, i.e., Berry and Tichy, while the phrase "the investigators" refers to Sobel and Clarkson.[1]

In the *TSE* paper, the investigators describe an experiment in which two groups of mostly two-person teams of university students were asked to develop running programs to meet the requirements of a given problem, an elevator simulation problem. One group of teams developed formal specifications, the other did not. The investigators observe that the formal methods group's solutions are "far more correct than the nonformal solutions." Additional details appear in a second paper, hereafter called the "Inroads paper," authored by only Sobel [4].

The formal methods group consisted of undergraduate students who had voluntarily participated in a formal methods curriculum. This curriculum consisted of a course on formal program derivation and a course on the axiomatic semantics of data structures, both taught using a first-order-logic specification language, plus a course on object-oriented design including UML. The other group, the control group, consisted of undergraduate students whose training differed in that they did not take part in the program derivation course, took a data structures course covering the same topics as the formal group except for the axiomatic semantics, and took the same course on OO-design. The elevator programming task was an assignment in the OO-design course. There were additional courses to be taken later.

Both curricula taught the same material, in the same sequence, by the same instructors, using the same examples, the same programming assignments, and the same exams, except for formal methods. Thus, the investigators have tried to maintain the equivalence of the two groups except for the experimental treatment, the continual exposure to formal methods by the members of the formal methods group.

The programming task used to assess the two groups was the development of an elevator simulation. Each group divided into teams, each with two members on average. Each team was to develop a running solution as a homework assignment. Six teams of the formal methods group and 11 teams of the control group handed in solutions that compiled properly.[2] Each team was encouraged to submit UML diagrams of its design. Each formal method teams was asked to submit a formal specification of its solution.

The investigators found that 100 percent of the programs produced by the formal method group teams passed all of a set of six test cases, while only 45.5 percent of the programs produced by the control group teams passed all of the same set of test cases. This is the main result of the experiment and seen as strong evidence of the power of formal methods.

Standardized ACT tests found no statistical difference between the abilities of students in the two groups at the beginning of the curricula. The investigators conclude, therefore, that the two populations, the students in the two groups, are alike in all aspects except for the training in formal methods.

1. This is a simplification because Clarkson was actually a student participating in the experiment.

2. There is a discrepancy regarding the number of formal methods teams—it is four in the Inroads paper but six in the *TSE* paper. In personal communication, Clarkson asserted that the correct number is six.

> The conclusion can be made that it was the use of formal analysis employed by the students during the development of their software solutions which caused the increase in the correctness of their solutions.

That is, the investigators conclude that it was the use of formal analysis by the formal methods group teams that caused that 100 percent of the programs produced by formal methods group teams passed all of six test cases while only 45.5 percent of the program produced by the control group teams passed the same six test cases.

3 Weaknesses of the Experiments

The experiment performed by the investigators had a number of weaknesses in its basic experimental design, its failure to recognize Hawthorne and novelty effects, its lack of control, and its poor measurement. These weaknesses reduce our confidence in its results.

3.1 Experimental Design

The major weakness of the experiment is that each student, by virtue of having decided whether or not to take the formal methods course, assigned him or herself to one of the two groups. Thus, assignment was voluntary and not random. The investigators assumed that the groups were nevertheless evenly matched on the basis of matching ACT scores and the fact that lectures presented the same material to the groups except for formal methods.

The design of the experiment is called *Nonequivalent Posttest-Only Design* in Christensen's book and is discussed in the section on **Faulty Research Design** (pp. 234-238).[3] As if to address the investigators' very experiment, Christensen explains that matching based on a number of variables such as age, sex, and ACT scores is "no assurance of having attained equated groups." Thus, it is ironic that the investigators announce that they have matched their groups based explicitly on ACT scores and implicitly on age (since all subjects were undergraduate students), all the while citing Christensen. However, Christensen writes:

> The only way one can have any assurance that the groups are equated is to assign subjects randomly to the two groups....In studies where it is not possible to assign participants randomly, the next best technique is to match on relevant variables. However, matching is no substitute for random assignments because it does not control for other variables *such as motivation* [emphasis added].

Nonequivalent groups do not provide a way of isolating the effect of the treatment condition because they cannot exclude rival hypotheses. There is strong evidence that the control and experimental groups in the experiment were affected by a number of extraneous variables, including the very motivation given as the Italicized such-as example by Christensen. At a bare minimum, the investigators should have alerted the reader to potential problems by classifying their work as a quasi experiment. A quasi experiment is one that has a design that does not meet all the requirements necessary for controlling the influence of confounding variables. Drawing causal interpretations from a quasi experiment is invalid, unless rival hypotheses arising from the confounding variables are shown to be implausible. In the following, we discuss the main confounding variables.

1. **Differences in motivation.** There are several clear indications that the experimental group was better motivated. The students in this group *volunteered* for a tougher curriculum: They were willing to take two extra courses, "Introduction to Program Derivation" and "Formal Analysis of Concurrent Programs."[4] Furthermore, the data structures course required more time than others because of the additional effort needed for studying and writing specifications. The Inroads paper says about the formal version of the data structures course (p. 159):

   > At times we were forced to meet outside of regularly scheduled class time. Some students found this pace gruelling.

   Only superior, well motivated students are willing to undergo such treatment. Page 158 of the Inroads paper reveals:

   > The [experimental] group remained highly motivated and committed to this educational experience.

   Finally, four of the formal methods students later volunteered to undertake a completely formal derivation of the program. The work was presented at the 1998 SIGCSE conference as a poster. The work shows that at least four students of the experimental group were highly motivated as well as highly talented. One of the formal methods students even became a coauthor of the *TSE* paper.
2. **Differences in exposure.** The experimental group received significantly more instruction. Students in this group took an extra course and had a tougher data structures course which required extra time to cover the material. It is likely that by having been taught longer and more carefully, they performed better in the programming task. Put another way, because the control group was not instructed as thoroughly, it is not surprising that its performance was lower.
3. **Differences in learning style.** The experimental group took a learning style survey that categorized the group members as collaborative and competitive; collaborative students share ideas among each other and competitive students want to receive recognition for their efforts. Since the control group members did not take the same test, it is impossible to exclude the rival hypothesis that subjects in the experimental group were better learners or harder workers.
4. **Differences in skills.** Students chose the formal or informal curriculum freely. The formal methods students knew that they were signing up for the tougher course sequence. ACT test scores not withstanding, we cannot exclude the hypothesis that the formal methods group were the more talented students.

   The Inroads paper contains indications that the experimental group did have superior analytic skills, as measured by a subset of the Computer Science GRE. We quote from page 160:

   > A general analytic exam was given to all senior majors in order to compare the two group's [sic] analytic skills when solving computer science problems. The questions were randomly selected from sample test questions in the analytic portion of the GRE examination....The formal methods students answered thirty-six percent more questions correctly than the average senior major.

   The GRE test was apparently given after the formal methods classes had been administered. It is unlikely that

3. It is not a before-after design because the pretest, the ACT, was different from the posttest, the grades in the programming class. Thus, it is not possible to perform a before-after comparison.

4. The second course was taken after the experiment concluded, but the students had volunteered for both.

> exposure to formal methods alone improved general analytic skills by over 30 percent. Thus, there is strong support of the rival hypothesis that the students in the formal methods class were the more talented students to begin with. The fact that ACT and exam scores did not differentiate the groups says something about the differentiating power of these tests, but not enough to discount the GRE results. It is unclear why the GRE test results were not mentioned in the *TSE* paper, especially since the Computer Science GRE test is more appropriate for assessing the quality of the students relative to the research question.

Given the differences among the groups enumerated above, a substantial number of rival hypotheses cannot be excluded. The experimental group could have outperformed the control group because of better motivation, harder work, more practice, more exposure, better learning style, or superior skills and talent.

Indeed, one of us, Berry, has hypothesized that it is more the nature of people who willingly and enthusiastically apply formal methods than the formal methods themselves that accounts for the reported successful applications of formal methods [1].

These willing and enthusiastic appliers of formal methods are, as Martin Feather observes,

> particularly adept at getting to the heart of any problem, abstracting from extraneous details, carefully organizing their whole approach to problem solving, etc. Surely, the involvement of such people would be beneficial to almost any project, whether or not they applied "formal methods."

Perhaps, in the investigators' study, the self assignment into the formal methods group caused the formal methods group to consist of people who were more adept than those in the control group at getting to the heart of the problem, abstracting from extraneous detail, and carefully organizing their whole approach to solving the problem. Perhaps these differences did not show up as differences in the standard tests. Indeed, it is not even clear how to test this sort of ability in short multiple choice questions given in typical standardized tests.

Random assignment of subjects to treatment would have helped avoid some of the above problems, but not all. Since the treatment lasted several semesters, students would have noticed sooner or later that they were in a particular curriculum because of a trial of formal methods and would have started behaving differently given this knowledge (more about the Hawthorne effect see in the next section). Thus, an alternative design would be needed. For testing program quality, *all* students should have been trained in formal methods. The teams should have been assigned randomly to a formal or informal solution just prior to the actual task (the posttest), for instance, by handing out different task descriptions according to a random number generator. We discuss several possible improved designs for this experiment in Section 3.4.

Note finally, that we are not blaming the authors for not having random assignment, since they also wanted to check problem solving skills with and without formal methods training. We are merely stating a fact. In this case, with the experiment conducted with university students and in which training for the treatment was delivered as a course, self assignment was unavoidable; students could not be forced to take this course. However, as necessary as self assignment may have been, it nevertheless introduces a confounding factor that cannot be separated out, namely, that the people in the formal methods group may naturally be better at systematic thinking, independently of whether or not they use a formal method. Thus, the favorable results may be at least partially a consequence of the formal methods people's natural abilities and not only of the formal methods.

3.2 Hawthorne Effect and Novelty Effect

The Hawthorne effect refers to the fact that subject performance is affected by the knowledge of being in an experiment. For instance, subjects knowingly participating in an experiment may be more willing to perform certain tasks than under normal conditions or may perform them more diligently; students in particular have a strong desire to please their instructors. The novelty effect states that subjects behave differently when asked to do something new or different. When the novelty wears off, the treatment effect might disappear as well. If the Hawthorne or novelty effects occur, the experimental results are not produced by only the treatment.

It seems quite clear that the students knew they were participating in a trial of formal methods and that this was a novelty. The Inroads paper states:

> The students involved [in the experimental group] formed a fraternal bond among themselves and their group identity extended well beyond the classroom. We have already scheduled a reunion ....

Furthermore, the Inroads paper reports some enthusiastic comments by the formal methods group, for example:

> Leads to something in academia that is applicable in the real world regardless of consciously or subconsciously. Awesome!

The expectation of the main investigator, Sobel, was quite clear. On page 309 of the *TSE* paper, we find:

> Given the goal of *establishing an increase*[emphasis added] in the complex problem solving skills of students who use formal analysis, ...

We find it unlikely that the goals of the experiment were kept from the students, given two parallel curricula with extra work in one of them, the goal of the grant, and the investigators' enthusiasm for formal methods. There is nothing wrong with experimenter expectations. In fact, it is unavoidable and legitimate for experimenters to have desires and biases. The problem arises when subjects behave differently because of their knowledge of experimenter expectations. They help, perhaps subconsciously, to fulfill the expectations. It is often extremely difficult to withhold experimenter expectations from subjects, and if one does, then subjects might guess them anyway. The only way to avoid that experimenter expectations affect the results is a blind or partially blind technique. For example, the investigators could have trained *all* subjects in formal methods, then let an independent proctor assign them randomly to formal or informal programming task at the time of the assignment. It is also possible to fully automate the test procedure to avoid experimenter-subject interaction. Furthermore, subject expectations and the positive self-presentation motive should have been neutralized. A possible approach is deception, i.e., giving the subjects a hypothesis unrelated to the real hypothesis. For instance, the test could have had no indication that there were different task prescriptions. Christensen provides several insights in this regard. Finally, a posttreatment questionnaire should have asked what the subjects thought of the experiment, what they thought the experimenters expected, and how they thought others would respond in this situation. Such a questionnaire helps separate out subject perceptions.

3.3 Lack of Control

A control group serves as a source of comparison. For this experiment, there is no record of what the control teams actually did. The *TSE* paper states that "almost no record of the design (if any) these teams used is available. None of these teams elected to submit a UML diagram, as they were invited to do." However, three of the six formal teams submitted UML diagrams; all of the formal teams submitted specifications, though to varying degrees of completeness.

Two rival hypotheses arise from this situation:

1. The control teams spent no or little time on the analysis of the problem and instead started programming early. This hypothesis springs from the well-known fact that students try to get by with the minimum effort. It is no surprise that the control group did not produce UML diagrams or written, informal specifications. The experimental group, however, was a highly motivated group and half of these teams produced the optional diagrams. If the hypothesis of a lack of informal specifications in the control group is true, it is not surprising that the resulting programs fared poorly.

   There is indirect support for this hypothesis because the Inroads paper reports that, even in the formal curriculum,

   > Some students reported actually "fighting" the overwhelming desire to start the problem solving process by coding.

   The investigators also noted that the control group delivered poor code: Four of the nine control teams produced tightly coupled designs, mixing display code with scheduling code, and several teams duplicated code. One control team duplicated the entire scheduling code for each elevator. These coding foibles are symptoms of poor or nonexistent designs. It is reasonable to assume that students not in the formal track had less encouragement to step back and analyze the problem, formally or informally, before coding. We conjecture that at least some of the teams in the informal group started problem solving with coding.
2. The control teams might actually have performed no analysis, informal analysis, formal analysis, or a mixture.[5]

The point is that too little is known about the behavior of the control group to serve as a source of comparison. The experiment compares the formal methods group with a group whose problem solving approach is essentially unknown. Since there was no supervision and no records were kept, one can argue both sides of the issue ad infinitum. In the end, the reader is left in doubt.

In controlled experiments, the behaviors of both the experimental and control groups must be observed closely.

1. In the present experiment, handing in designs, formal or informal, depending on the group, should have been mandatory. For formal method application to have *caused* the benefits claimed for them, the application had to have been carried out *before* coding began. We find no clear statement as to the order of the activities. For example, one might expect some statement about the students having to submit specifications, designs, and running programs for grading in separate homeworks due in that order. On the other hand, in the *TSE* paper's Section 7.2, we see a description of a fully formal solution carried out after the experiment was finished and, in this solution, the formal specification was completed before the implementation began.
2. Furthermore, all subjects should have worked in a programming environment with automatic capture of the majority of actions, for example, start and finish times, the times at which subjects compiled, debugged, edited, etc. The various work products should have been captured and stored for later analysis.
3. Finally, instead of allowing the students to work unsupervised, the students should have worked in a laboratory in which an impartial proctor could have supervised and ensured that the desired behaviors were followed.

5. This hypothesis was pointed out by one of the investigators.

Up to this point, we have argued that there are a number of alternative hypotheses arising from the quasi-experimental design. Since these have not been shown to be implausible, it is not valid to draw the causal conclusions that the experimenters advocate. We now turn to the question of whether measurements were properly taken and analyzed.

Please read!

3.4 Analysis of the Results

In a paper about experiments, one first sees a statement of a general research question, e.g., "We study the effects of formal methods on the production of correct software." Then, she sees one or more concrete testable hypotheses, typically stated in form of a null hypothesis, e.g., "There is no difference in the correctness of programs produced by application of formal methods from that of programs produced without application of formal methods." Then, the paper goes on to reject or fail to reject each null hypothesis based on data gathered during the experiment.

The introduction and conclusion of the *TSE* paper refer to two different benefits (here, we quote from the introduction):

> Formal methods students had increased complex problem solving skills.
>
> The use of formal analysis during software development produces "better" programs.

These research questions can be stated as two hypotheses, assuming that the terms are defined to be measurable.

1. Teaching formal methods to undergraduate students increases their skill to solve complex problems.
2. The use of formal analysis during software development produces programs that are more correct.

Hypothesis 2 is indeed captured in the first paragraph of the *TSE* paper's Section 3 (we quote):

> ...this experiment tests the hypothesis that the formal methods group solutions were better than the control group solutions (using the criteria of code correctness, conciseness, and complexity) due to their use of formal methods.

In giving the hypothesis, the investigators give also a way to measure the goodness of solutions, via code correctness, conciseness, and complexity.

These two hypotheses require different experimental designs. When testing the first hypothesis, the control group should have been taught alternative material to compare against because teaching the same material more thoroughly can also be done with approaches other than formal methods. In particular, the control group could have been trained in the use of informal, but precise specifications. Comparing "nothing" with formal methods leads to strongly biased results, as explained above. Also, problem solving skills, even if limited to programming, could be tested more directly than by requiring a running program. The problem is that delivering a running program requires skills besides problem solving, for example, teamwork. Although one might be tempted to see teamwork skills as part of problem solving skills, it is unclear how formal methods affect teamwork skills.

The design of the investigators' experiment addresses the second hypothesis, regarding the quality of programs. The investigators compare several software metrics and find no significant differences. However, on functional correctness, measured by running six test cases, the "formal method group's solutions are found to be far more correct than the nonformal solutions." All six implementations of the formal group passed all six test cases, but only five out of the 11, or 45.5 percent, of the implementations of the informal group passed the same test cases.

The investigators did not provide a test for significance of the difference in functional correctness. Since the distributions appear strongly nonnormal, the Mann-Whitney U-Test for the difference

of the medians should be used rather than a parametric test. We reconstructed the test data to the best of our knowledge and found the difference to be indeed significant at the $p = 0.05$ level. However, is this statistically significant difference meaningful? The trouble is the size of the test case set. It consists of only six test cases. Nothing is revealed about the test cases themselves. Even though the average length of the programs is only 129 lines, six test cases are hardly sufficient to provide a representative sample of the state space of something as intricate as elevator scheduling. The measured superiority of the formal solutions could be entirely due to the choice of test cases. In other words, the data points and, hence, the given percentages are untrustworthy. The 5 percent p-value of the significance test says merely that, if one tests elevator schedulers with *exactly those six test cases*, then a result different from that observed might occur in less than 5 percent of the repetitions of the experiments. We know nothing at all about what would happen in a more realistic test scenario.

It is surprising to learn that the main result of the paper rests on a set of six test cases. This situation is particularly distressing because the elevator problem is amenable to automatic test case generation. The appropriate approach would have been to randomly generate a large number of test cases that sample the state space of the elevator problem uniformly. Of course, manually generating a large number of test cases, determining their correct answers, and comparing them with the outputs of the programs is infeasible. However, the outputs of the programs could have been compared by software with those of a so-called gold program. An observably good program developed by one of the formal methods teams would have been a good candidate for such a gold program. Alternatively, the program of the fully formal solution carried out after the experiment was finished would make the perfect gold program. A testing framework such as CppUnit could have been used to automate the testing process.

In conclusions, significance tests or even simple ratios, provided by an experimenter are only as good as the quality of the data on which they are computed.

Finally, we point out a major missed opportunity, that of observing the time students spent on their solutions. An important question is whether the use of formal methods requires an increase in time or not. Any rational programming method can be made to produce good results, if the program is small enough and subjects work hard enough at it. Thus, for useful results applicable to real-life software development, it is mandatory to explore not only program quality but also the time required to achieve the observed quality.

3.5 Related Work

We were surprised that the earlier study of formal methods by Pfleeger and Hatton [3] was not cited. The study is directly related to the second hypothesis and comes to a contradictory result. Thus, it is important for the investigators to explain the differences. For instance, could the differences be caused by the different settings, i.e., students working on homework versus professional programmers working on safety-critical software? Although the Pfleeger and Hatton work is a case study rather than a controlled experiment, its results are quite strong, because it uses data from a real and sizeable system, an air-traffic-control information system of about 200,000 lines. The study reports on over 3,000 fault reports before delivery and 273 after delivery generated by an unknown but large number of tests, for a span of over four years. Pfleeger and Hatton compared modules that were developed informally with those that were developed using finite state machines, VDM, or CCS. Pfleeger and Hatton

> ...found no compelling evidence that formal design techniques *alone* produced code of higher quality than informal design techniques.... We can conclude that formal design, combined with other techniques, yielded highly reliable code.

In the face of such prior work, at least a brief discussion of the differences is necessary.

3.6 External Validity

A research report on an experiment should include an honest and careful discussion of threats to construct, internal, and external validity. Only if shortcomings are discussed can readers decide where the experiment was strong, where it was not, and where additional research is needed. There is no discussion of threats to validity in either the *TSE* paper or the Inroads paper, although the Inroads paper is quite candid about what actually happened. Authors should resist the temptation to present a sanitized version of an experiment. Occasionally, we have encountered fears among experimenters and reviewers that discussing problems would distract from the main message, weaken the results, even throw the whole experiment into doubt. The opposite is the case. Remaining silent about threats to validity raises suspicion among experienced readers because they know that no experiment is perfect. Experiments interact with the real world and unexpected or unavoidable problems arise. Experimenters, reviewers, and readers need to accept that experiments do not occur in an ideal or theoretical world. When problems surface, careful judgments have to be made about whether the problems invalidate the results. Making such judgments, however, requires that the report reveal and discuss those flaws that may have had a significant influence.

4 Conclusion

Perfection in experiments, especially in those involving human subjects, is unattainable. In the Sobel and Clarkson experiment, the many problems mentioned in this note call for a complete redesign of the experiment. Clearly, the research question is immensely important. Given the sizeable research and teaching investment that has gone into formal methods worldwide for over three decades, extensive studies are warranted that analyze whether the precision, structure, and discipline of formal methods do indeed improve problem solving skills, program quality, or programmer productivity. Sobel and Clarkson deserve the credit of a first attempt at answering these questions with a quasi experiment. We hope that, by studying the problems cited in this note, future experimenters can avoid these problems and gather data from which valid, meaningful, and significant conclusions can be drawn.

Acknowledgments

The authors would like to thank Mike Godfrey, Matthias Müller, Karl Reed, and Steve Schach for their comments on an earlier draft of this paper. They also thank Bill Berry for a careful explanation of certain statistical issues in experiment design. Any mistakes remaining after this explanation are solely our faults. D.M. Berry was supported in part by NSERC grant NSERC-RGPIN227055-00.

References

[1] D.M. Berry, "Formal Methods, the Very Idea, Some Thoughts on Why They Work When They Work," *Science of Computer Programming*, vol. 42, no. 1, pp. 11-27, Jan. 2002.

[2] L.B. Christensen, *Experimental Methodology*, eighth ed. Allyn and Bacon, 2001.

[3] S.L. Pfleeger and L. Hatton, "Investigating the Influence of Formal Methods," *Computer*, vol. 30, no. 2, pp. 33-43, Feb. 1997.

[4] A.E.K. Sobel, "Empirical Results of a Software Engineering Curriculum Incorporating Formal Methods," *ACM Inroads*, vol. 32, no. 1, pp. 157-161, Mar. 2000.

[5] A.E.K. Sobel and M.R. Clarkson, "Formal Methods Application: An Empirical Tale of Software Development," *IEEE Trans. Software Eng.*, vol. 28, no. 3, pp. 308-320, Mar. 2002.