N = 1: *an alternative for software engineering research?*[§]

Warren Harrison
Department of Computer Science
Portland State University
Portland, OR 97207-0751
USA
503-725-3108
warren@cs.pdx.edu

Abstract

The "standard" approach to studies in empirical software engineering research is the "comparative group experiment". We are told the participation of large groups of subjects makes the results more conclusive and meaningful. However, requiring large numbers of subjects results in issues involving the expense, timeliness and applicability of results. Many of these can be addressed through the use of a technique long used in psychological research: *the single subject experiment*. In Single Subject experiments, a single subject is systematically studied and the factor of interest is alternatively introduced and withdrawn so its effect on the subject can be analyzed.

Keywords: controlled experimentation, single-subject experiments

Introduction

One of the primary paradigms for research in empirical software engineering tends to be the comparative group experiment. Usually, this entails two groups of subjects performing some task - the "experimental group" and the "control group". One group benefits from the inclusion of the technique or artifact being studied (the "experimental variable"). The other group typically performs the task in the absence of the experimental variable. The performance of the two groups is subsequently compared to assess the impact of the experimental variable. One pretty much universally observed rule is that larger (i.e., more subjects) experiments carry more weight than smaller experiments. The ramifications of requiring "Big-N studies" in order to draw meaningful conclusions presents certain problems to the empirical community:

- *Over-Reliance on Student Subjects*. The drive for large-N experiments may have resulted in an over-reliance on student subjects. Professional programmers are hard to come by and are very expensive. Thus, any study that uses more than a few professional programmers must be very well funded. Further, it is difficult to come by an adequate pool of professional developers in locations that do not have a significant

[§] This paper is based upon an editorial of the same title in Volume 2, Number 1 of *Empirical Software Engineering* (1997)

software development industrial base. Even if we can somehow gather a sufficiently large group of professionals, the logistics of organizing the group into a set of experimental subjects can be daunting due to schedule and location issues. On the other hand, students are cheap, plentiful, and easily managed. However, it is unclear how well results from student-based experiments generalize to professional software engineers. Clearly, students (even "advanced graduate students") behave differently than professionals, yet virtually every student-based study attempts to generalize their results to the industrial environment.

- *Problems in "Task Realism".* Software Engineering experiments ordinarily focus on activities that strain the intellectual capacity of experienced professionals. For instance, developing a piece of complex code, tracking down a programming error, or analyzing an architectural design. However, it is difficult to manage and control a large number of subjects (even students) for long periods of time. For instance, it would be difficult to ensure that members in one group of subjects would forego communicating with the members of another group of subjects for an extended period of time. However, such interaction might very well ruin the design of an otherwise well constructed experiment. When dealing with students, the length of an experiment is almost always limited to artificial constraints due to academic periods. Consequently, the preference is towards short-term experimental tasks. For instance, typically the experimental variable must yield an observable effect in a matter of hours rather than six months or a year. Even "long term" studies will usually be limited to three to five months (the typical period of an academic term). On the other hand, many factors that we may wish to study require a significant period before obtaining meaningful results. For instance, the benefits claimed for many process improvements are reduced rework costs six months to a year later in the lifecycle. The effects may be imperceptible, or even negative if we attempt to measure them too soon. Other factors possess a fairly steep learning curve that must be surmounted before the benefits begin to accrue. A study that introduces a factor such as inspections, has subjects apply it, and attempts to measure the benefits within a one month period (while the subjects are also juggling three other classes, homework, etc.) are likely to be either inconclusive or just downright wrong.

- *Unrealistic Environment.* Even when subjects do perform realistic tasks, they may be carried out in an unrealistic manner. Professional developers usually work within a sophisticated infrastructure consisting of workstations, software tools and physical plant. For instance, a commercial software engineer will likely have at least a private cubicle, modern workstation, a collection of commercial software tools and professional staff support. In most academic research settings, it would be a challenge to get this kind of support for a research assistant (or in some cases, it may be difficult to get this kind of support for the researcher himself). It is indeed the rare situation in which an experimental lab can be equipped to provide this kind of environment for a Big-N experiment. The effect of such an infrastructure (or lack thereof) on laboratory experiments can only be guessed, but certainly it impacts the results.

- *Inappropriate Grouping of Subjects.* The increased "power" of large-N studies may be misleading. We know that ability varies among programmers by an order of magnitude or more. While the issue of "ability" has at least been raised within our community, it is unclear that factors such as "cognitive style", or differing approaches to problem solving have even been considered. We know that some individuals prefer to view their data graphically while others prefer to see it in tabular form. Some techniques may work well for novice programmers but just get in the way of more experienced ones. Typically Big-N studies aggregate subjects of many different levels of experience, cognitive style, ability, etc. in order to achieve their "Big-N". While most studies are careful to ensure that experimental groups are homogeneous - i.e., they represent similar mixes of experience, ability, etc., they tend to miss the point that by mixing such individuals (even if the mixture is homogeneous between groups) the affect of the treatment will be lost. For example, if a group of subjects is comprised of equal numbers of individuals with and without a "visual" cognitive style, the benefits of a visual debugging tool may very well "average out" to neutral. In reality of course, we might find that such support is a boon to those with a visual cognitive style, but impedes those who prefer to "see their data in columns". These conclusions are radically different than the "no effect" result we might get if we average the performance out over the group.

Many of these limitations may be mitigated through huge infusions of funding. For instance, given enough resources, an institution located in an area with a significant software development industrial base could almost certainly attract adequate professional subjects from "hi-tech temp services". These subjects would not be constrained by traditional academic calendars, and complete double blind designs could be enforced. Special labs emulating an industrial development environment could most certainly be constructed with adequate resources (of course, we must ensure that the laboratory infrastructure keeps current, so we also need to budget for on-going maintenance and upgrades). Tests exist by which to identify an individual's cognitive style, and if we are sensitive to the issue and have a large enough pool of subjects from which to draw, we could construct representative groups of subjects.

Single Case Experimentation

Clearly, given adequate resources, these limitations can be addressed. However, such funding for experimental software engineering is not on the horizon. Yet these issues have serious ramifications on much of our work. Fortunately, a well-developed solution to these problems can be found in the literature that has sprung up around single case experimental design [1,2], particularly in the fields of psychology, psychiatry and education. To quote Skinner [3]:

> *"... instead of studying a thousand rats for one hour each, or a hundred rats for ten hours each, the investigator is likely to study one rat for a thousand hours."*

The A-B Design

Single case experimental designs usually involve an initial period of observation comprised of repeated measurements of the dependent variable. This initial period is referred to as the "baseline", and is intended to provide a standard by which manipulations of the experimental variable can be compared. Typically this period continues until a stable pattern emerges. Upon completion of the baseline (or "A") phase, the independent variable is manipulated (the "B" or "treatment" phase), at which point changes in the dependent variable are noted. This procedure is known as an A-B design.

Consider a simplified example involving a symbolic debugger. During an initial period, the subject develops software without the aid of a debugger. During this baseline phase data on effort spent tracking down and fixing bugs is recorded. Over a period of time, the performance of the subject will "stabilize" and we can conclude that this subject spends a certain percentage of his time "debugging". We can then introduce a symbolic debugger and collect additional performance data. At some point the performance measures collected during the stabilized Baseline (i.e., "A phase") are compared with the performance measures collected during the treatment phase (i.e., the "B: phase).

Such a study can accommodate many of the issues raised earlier with respect to Big-N controlled experimentation. While not cheap, a single professional can be obtained more easily and at a far lower cost than a group of 30 or 40. An infrastructure can be assembled much more easily for one programmer working for several weeks or months than 30 programmers working for only a single hour. Likewise, because the baseline, training and treatment phases may each take several weeks, the study can overcome the problems of expecting too much too soon and getting too little too late. Finally, the subject represents his or her characteristics in the study. Results can only be generalized to other programmers of similar makeup. Clearly, in return for reducing the burden of acquiring a large number of subjects, the researcher can expect a lengthy study, consisting of weeks, or even months.

A-B-A-Withdrawal Design

Of course, it may be difficult to substantiate the effects of the treatment using the classical A-B design. For instance, how can we be sure the change in performance when the debugger is introduced is truly due to manipulating the experimental variable? Because of this, an elaboration to the A-B design known as the "A-B-A withdrawal design", is in use. In the A-B-A withdrawal design the treatment is removed, and another baseline period entered. At the conclusion of this second baseline period we should have data that further explores the effect of the debugger on the programmer's debugging effort. If the debugger was indeed responsible for improved performance, then removing it should result in a corresponding *decrease* in performance. Obviously an A-B-A withdrawal design will only work with treatments which can be reasonably removed, such as a programming tool or access to specific information. Skills and techniques once learned are difficult, if not impossible to unlearn.

Variations of the A-B-A design include cycles of B-A periods (i.e., A-B-A-B-A-B), so the method can be likened to exercising a control knob attached to a piece of machinery and verifying the connections are working.

Challenges for Single Subject Experimentation

Naturally, single subject experimentation has many issues associated with it, which must be overcome. These issues tend to deal with statistical limits, the generalizability of results and the problem of measurement.

Statistical Limits for Single Subject Experiments

Even with the A-(B-A)* improvements over the simple case study, the single subject technique still falls victim to the issue of statistical analysis. Most tests for statistical differences between groups used with traditional experimentation are based on means and standard deviations. However, such concepts are not usable with a single subject experiment. Thus, most single subject experimenters subscribe to the concept of a *therapeutic criterion*. Plainly put, this refers to whether the effects of the treatment are important or not. Even if a change can be reliably attributed to the treatment, it simply may not be of practical significance. To achieve the therapeutic criterion, the treatment needs to make an important change in the subject's effectiveness.

The therapeutic criterion is grounded in the philosophy that for a technique to be adopted, it must result in significant changes in behavior. These changes must be so significant, that statistical hypothesis testing is unnecessary to recognize a significant difference due to a treatment. Given the assumption that any improvements worth noting must be meaningful, simple visual analysis of data is quite popular among single subject researchers since the impact is expected to be relatively large. However the use of randomization tests and time-series analysis are also used within this community.

Generalizablity of Single Subject Experiments

Perhaps the most contentious objection to single subject research is that it is unclear if "treatments" so evaluated are generalizable. This objection is usually addressed through the use of replication in which additional single subject studies, following the same procedures and manipulating the same independent variables, are carried out. Assuming the procedure is reliable, the therapeutic criterion is met, and the subjects in the replications are homogeneous, it would be expected that similar results should be observed in every case. On the other hand, it is important to understand, the population for which the study generalizes is represented by the subject himself and no one else. Differentiating among the subject and the rest of the software engineering labor pool can be a challenging task.

Measurement Issues

As in any other kind of experimentation, the analysis must be based upon well founded, consistent, and clearly defined measures. For instance, in the earlier debugger example, the performance measure was the time spent debugging. However, even something this apparently simple is not what it seems. Should time be normalized by the amount of

software being developed or number of bugs found? To what level of granularity should time be measured? Should the measurement be assessed in terms of ordinal categories such as "between thirty minutes and one hour", or actual minutes? Because of the reliance of single subject experiments on the therapeutic criterion it may well be that measurement issues are even more important than in more traditional studies.

The Relationship of Single Subject Experiments to Case Studies

At first glance, this procedure appears much like a case study. However, the A-B-A withdrawal design corrects for the deficiencies of the traditional case study in two important ways:

> (1) A well run single subject experiment imposes rigid controls on measurement of the dependent variable and changes in the independent variable. To ensure the treatment responsible for the change is recognized, only one variable is changed at a time. This increases our confidence that a change in the dependent variable actually occurred and that it was due to the manipulation of the specific independent variable. Obviously in case studies, we take the subjects as we find them, and it is never clear which treatments are responsible for which performance changes.
>
> (2) The use of the withdrawal phase adds an element of prediction to the study. If indeed, the introduction of a debugger improved the programmer's effectiveness, then its removal should result in a concomitant reduction in the programmer's effectiveness. In case studies, it is usually impractical to re-run the case study and modify one of the variables.

While at first glance, it may appear that a single subject experiment is nothing more than a case study, there are substantial differences. These differences are intended to help identify and isolate the variables affecting the activity.

Conclusions

Single subject experiments can address many of the issues from which classical "comparative group experiments" suffer. We can gain a great deal in learning about the use of single subject experimentation in other domains such as psychology, education and psychiatry.

References

[1] Barlow, D. and M. Hersen, *Single Case Experimental Designs*, Pergammon Press, 1984.

[2] Kratochwill , T. and J. Levin (editors), *Single Case Research Design and Analysis*, Lawrence Erlbaum Associates, 1992.

[3] Skinner, B.F., *Operant behavior: Areas of research and application*, Appleton-Century-Crofts, 1966.