

Skinner Wasn't a Software Engineer

Warren Harrison

Most of us have heard of B.F. Skinner (see www.bfskinner.org), the father of operant conditioning. If you recall, Skinner pioneered those psychology experiments that we often hear about where a rat is placed in a box and learns to press a lever a certain number of times in order to receive a pellet of food.

Even before Skinner, Ivan Pavlov (of Pavlov's Dogs fame) studied behavioral conditioning in dogs. Every time he fed his dog, he rang a bell. After a while, his dog would start to salivate when he heard the bell whether food was present or not.



We won't be duped

Of course, the software engineering community would never fall for the work by Skinner, Pavlov, or the many other psychologists who've studied operant conditioning. After all, each of these studies only included a handful of animals. We all know that to prove something significant, you need a large number of subjects. Otherwise, how do you know you're not working with an extraordinary beagle or pigeon? Perhaps most rats just can't learn, and you just happened to pick the one genetic mutant that has a more developed thalamus.

Software engineering researchers and practitioners are all familiar with one of the main approaches to arguing the value of a new tool or technique: the comparative group experi-

ment. Typically, this involves comparing the performance of two groups of subjects, one that employs the new tool or technique and one that does not. For instance, if we're investigating a new development paradigm, we might study the members of each group as they develop or modify a software application. Members of one group use the new development paradigm, while members of the other don't. Upon completing the task, we'd compare the two groups' average performance. Did members of one group complete the task more quickly or make fewer mistakes than the other?

These experiments are much more convincing if the groups consist of dozens of programmers rather than three or four. When we have very small group sizes, random occurrences can significantly affect the outcome. For instance, if we have only two programmers in a group and one of them is suffering from the flu, their group performance probably depends more on fever than on development paradigm. On the other hand, if the group has 100 programmers, the other 99 will mitigate the impact of a single programmer performing below his normal abilities.

But where do I get 100 programmers?

As you can imagine, finding 100 developers willing to participate in such an experiment is neither cheap nor easy. Even a modest experiment could cost tens of thousands of dollars. But even if a researcher has the money, where

DEPARTMENT EDITORS

Bookshelf: Warren Keuffel,
wkeuffel@computer.org

Design: Martin Fowler,
fowler@acm.org

Loyal Opposition: Robert Glass,
rglass@indiana.edu

Open Source: Christof Ebert,
christof.ebert@alcatel.com

Quality Time: Nancy Eickelmann,
nancy.eickelmann@motorola.com,
and Jane Hayes, hayes@cs.uky.edu

Requirements: Suzanne Robertson,
suzanne@systemsguild.com

Tools of the Trade: Diomidis Spinellis,
dds@aubg.edu

STAFF

Senior Lead Editor
Dale C. Strok
dstrok@computer.org

Group Managing Editor
Crystal Shif

Senior Editors
Shani Murray, Dennis Taylor, Linda World

Staff Editor Editorial Assistant
Rita Scanlan Brooke Miner

Magazine Assistant
Hilda Hosillos, software@computer.org

Art Director
Toni Van Buskirk

Technical Illustrator
Alex Torres

Production Editor Production Artist
Monette Velasco Carmen Flores-Garvey

Executive Director
David Hennage

Publisher
Angela Burgess
aburgess@computer.org

Assistant Publisher
Dick Price

Membership/Circulation Marketing Manager
Georgann Carter

Business Development Manager
Sandra Brown

Senior Production Coordinator
Marian Anderson

CONTRIBUTING EDITORS

**Anne Lear, Robert Glass,
Molly Mraz**

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society's Web-based system, Manuscript Central, at <http://cs-ieee.manuscriptcentral.com/index.html>. Be sure to select the right manuscript type when submitting. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

do they find that many programmers? Resourceful academics often address this problem by pressing students into service. Students are cheap, plentiful, and easy to manage. Unfortunately, practitioners are understandably skeptical of results acquired from a study of 18-year-old college freshmen.

How long have I got?

Comparative group experiments tend to work best when outcomes are observable within a short period of time. This is because of both the cost as well as the ability to control the subjects. If your experiment takes an hour to carry out, you have some control over your subjects. You can ensure that they don't discuss the experiment among themselves, prevent them from using external reference material that might bias the results, keep close measurement of how much time they spend thinking about the problem, and so on. However, if the experiment takes several days, you really can't control for these factors short of sequestering the entire lot of them.

Consequently, many tasks are constructed to be exceedingly simple in order to squeeze into this artificial window of time. While this might be appropriate for some studies, many other problems require long-term study before obtaining meaningful results. For example, it's hard to see the results of a process improvement technique in an hour and a half.

Will field studies fix it?

A common alternative to the controlled group experiment is the field study—an analysis of the measurements taken in the process of developing a real software system. However, a field study's results can often be as misleading as academic studies involving students performing simple programming tasks.

There's still pressure to include a large number of data points. So, to achieve a large data set, we usually end up with a collection of projects developed using different processes and personnel and coming from different application areas. Often, we limit measures to phases

in which data collection can be expedited. For instance, many industrial studies measure defects found during integration testing because data is usually easy to extract from a defect tracking system. On the other hand, because projects seldom formally track errors found during unit testing, we almost always omit these errors from field studies.

n = 1?

Clearly, striving for large data sets has drawbacks, regardless of the context. As I observed in a paper at the Workshop on Using Multi-disciplinary Approaches to Empirical Software Engineering Research in June 2000 ("N = 1: An Alternative for Software Engineering Research?"), it might be more useful to consider people and projects as individuals rather than as single data points in larger collections of data points. Other disciplines have encountered similar issues. Clinical psychology and psychiatry have addressed these problems through single-subject experimental design. To quote Skinner (in *Operant Behavior: Areas of Research and Application*, Appleton-Century-Crofts, 1966):

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.

Like comparative group experiments, single-subject experiments involve a treatment—the new technology under study—as well as a measure of performance. However, unlike the group experiment, we study only one subject at a time. For software development studies, this might be a single developer, a single team, or a single project.

The single-subject experiment begins with a hypothesis. This is important because the hypothesis determines everything else that follows, from defining "the treatment" to deciding which performance measures to use.

We start our data collection with an initial period of observation, called the *baseline*. We record the performance measures before applying the treatment.

Called the A Phase, this can last days, weeks, or months. The point of the A Phase is to provide a standard against which we can compare performance once we apply the treatment.

Once we establish a performance baseline, we apply the treatment; then, we measure the performance again. We call this the B Phase. For instance, if our treatment involves integrating a lint tool into our build procedure, we'll compare the programmer's error rate after introducing the tool to the A Phase error rate. Any changes we see in the programmer's performance are likely (though not necessarily) due to the treatment. (However, if some confounding factor exists, we're much more likely to recognize it with a single programmer than with a group of 30.)

We call this an A-B design. However, it would be premature to assume the improvement in error rate is due to the lint tool's introduction. Therefore, the single-subject experiment involves a third phase, in which we withdraw the treatment. For instance, we could remove access to the lint tool and again measure programmer error rate. If the reduction in error rate really were due to the lint tool, we'd expect the error rate to increase during this third phase.

Adding the third phase gives us the A-B-A withdrawal design, and we can imagine additional elaborations, such as A-B-A-B, A-B-A-B-A, and so on. These are quite effective at ensuring that the performance changes we observe are due to the treatment under study and not an unanticipated external event.

What about statistics?

It's difficult to substantiate with classical statistical analysis that the treatment is responsible for a performance change when we use the A-B-A design. Statistics used in comparative group experiments depend on representative values such as means and medians and measures of variability such as standard deviations. Obviously, if we're measuring a single subject's performance, then means, medians, and standard deviations don't make much sense.

To overcome these issues, we evalu-

ate single-subject studies by determining whether the treatment's effects are noticeable. We call this the *therapeutic criterion*. To achieve this, a treatment must make an observable change in the subject's effectiveness that doesn't require elaborate statistical analysis.

Aren't single-subject experiments just case studies?


Case studies are popular for relating your experiences with introducing new software development technologies into an organization. A well-done case study shares some similarities with the A-B design. It obtains a performance baseline before starting, compares it to the performance observed at the study's end, and evaluates the outcome using the therapeutic criterion. Nevertheless, the typical case study differs from a single-subject experiment in a number of ways.

Usually, a case study is uncontrolled and establishes its hypothesis after the fact, whereas a single-subject experiment is hypothesis driven and controlled. Also, we can alternately introduce and remove the treatment in a single-subject experiment's withdrawal phase, giving us an idea of the treatment's significance.

Does anyone use single-subject experiments?

To date, single-subject experiments to study software development advances are almost nonexistent. Obviously, not every study lends itself to this approach—some treatments aren't easy to withdraw. However, single-subject experiments can be a powerful tool in both the researcher's and practitioner's toolkit.

Feedback welcome

I'd like to find out what you think. What does it take to convince you that a new technique or method is valuable? What does it take to get you to try out a new technique? Have you or anyone you know had any success in using single-subject experimentation? Please write me at warren.harrison@computer.org. 

EDITOR IN CHIEF

Warren Harrison

10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314
warren.harrison@computer.org

EDITOR IN CHIEF EMERITUS:
Steve McConnell, Construx Software
stemcco@construx.com

ASSOCIATE EDITORS IN CHIEF

Education and Training: Don Bagert, Rose-Hulman Inst. of Technology; don.bagert@rose-hulman.edu

Design: Philippe Kruchten, University of British Columbia; kruchten@ieee.org

Requirements: Roel Wieringa, University of Twente; roelw@cs.utwente.nl

Management: Don Reifer, Reifer Consultants; dreifer@earthlink.net

Quality: Stan Rifkin, Master Systems; sr@master-systems.com

Experience Reports: Wolfgang Strigel, QA Labs; strigel@qalabs.com

EDITORIAL BOARD

Christof Ebert, Alcatel
Nancy Eickelmann, Motorola Labs
Martin Fowler, ThoughtWorks
Jane Hayes, University of Kentucky
Warren Keuffel, independent consultant
Neil Maiden, City University, London
Diomidis Spinellis, Athens Univ. of Economics and Business
Richard H. Thayer, Calif. State Univ. Sacramento

ADVISORY BOARD

Stephen Mellor, Mentor Graphics (chair)
Maarten Boasson, Quaerendo Invenietis
Robert Cochran, Catalyst Software
Annie Kuntzmann-Combelles, Q-Labs
David Dorenbos, Motorola Labs
Juliana Herbert, ESICenter UMINSINO
Dehua Ju, ASTI Shanghai
Gargi Keeni, Tata Consultancy Services
Tomoo Matsubara, Matsubara Consulting
Dorothy McKinney, Lockheed Martin Space Systems
Bret Michael, Naval Postgraduate School
Susan Mickel, Lockheed Martin
Ann Miller, University of Missouri, Rolla
Deependra Moitra, Infosys Technologies, India
Melissa Murphy, Sandia National Laboratories
Suzanne Robertson, Atlantic Systems Guild
Grant Rule, Software Measurement Services
Girish Seshagiri, Advanced Information Services
Martyn Thomas, Praxis
Rob Thomsett, The Thomsett Company
Laurence Tratt, King's College London
Jeffrey Voas, SAIC
John Vu, The Boeing Company
Simon Wright, SymTech

CS PUBLICATIONS BOARD

Michael R. Williams (chair), Michael R. Blaha,
Mark Christensen, Roger U. Fujii, Sorel Reisman,
John Rokne, Bill Schilit, Linda Shafer,
Steven L. Tanimoto, Anand Tripathi

MAGAZINE OPERATIONS COMMITTEE

Bill Schilit (chair), Jean Bacon, Pradip Bose,
Doris L. Carver, Norman Chonacky,
George Cybenko, John C. Dill, Frank E. Ferrante,
Robert E. Filman, Forouzan Golshani,
David Alan Grier, Rajesh Gupta, Warren Harrison,
James Hendler, M. Satyanarayanan