4-7-2005

Experimental Program Analysis: A New Program Analysis Paradigm

Joseph R. Ruthruff
University of Nebraska - Lincoln, ruthruff@cse.unl.edu

Sebastian Elbaum
University of Nebraska - Lincoln, selbaum2@unl.edu

Gregg Rothermel
University of Nebraska - Lincoln, grothermel2@unl.edu

Technical Report TR-UNL-CSE-2005-0001, Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, Nebraska, USA, 7 April 2005

Experimental Program Analysis: A New Program Analysis Paradigm

Joseph R. Ruthruff, Sebastian Elbaum, and Gregg Rothermel
Department of Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, Nebraska 68588-0115, USA
{ruthruff, elbaum, grother}@cse.unl.edu

10 September 2005

Abstract

Program analysis techniques analyze software systems to collect, deduce, or infer information about them, which can then be used in software-engineering related tasks. Recent research has suggested that a new form of program analysis technique might be created by incorporating characteristics of experimentation into analyses. This paper reports the results of research exploring this suggestion. Building on background in classical experimentation, we provide descriptive and operational definitions of experimental program analysis, illustrate them by examples, and describe several differences between experimental program analysis and classical experimentation. We present three studies that show how the use of the paradigm can help researchers identify limitations of analysis techniques, improve existing experimental program analysis techniques, and create new experimental program analysis techniques by adapting existing non-experimental techniques.

Keywords: experimental analysis, program analysis, experimentation

1 Introduction

Program analysis techniques analyze software systems to collect, deduce, or infer specific information about those systems. The resulting information typically involves system properties such as data dependencies, control dependencies, invariants, anomalous behavior, reliability, or conformance to specifications. This information supports various software engineering activities such as testing, fault localization, impact analysis, and program understanding.

Zeller [21] describes a hierarchy of four reasoning techniques that can be utilized by program analyses. One of these techniques is *experimentation*, and he suggests that a new form of program analysis technique might be created by incorporating some of its characteristics. Such *experimental program analysis* techniques might be able to draw inferences about the properties of software systems in cases in which more traditional analyses have not succeeded.

We find Zeller's suggestion intriguing, because as experimentalists, we do indeed recognize many characteristics of classical scientific experimentation that already are, or could potentially be, utilized by program analysis techniques. These characteristics include the formulation and testing of hypotheses, the iterative process of exploring and adjusting these hypotheses in response to findings, the use of sampling to cost-effectively explore effects relative to large populations in generalizable manners, the manipulation of independent variables to test effects on dependent variables

while controlling other factors, the use of experiment designs to facilitate the cost-effective study of interactions among multiple factors, and the employment of statistical tests to assess results.

Anyone who has spent time debugging a program will recognize the relevance of several of the foregoing characteristics to that activity. Debuggers routinely form hypotheses about the causes of failures, conduct program runs (in which factors that might affect the run other than the effect being investigated are controlled) to confirm or reject these hypotheses, and based on the results of this "experiment", draw conclusions or create new hypotheses about the cause of the fault. The "experimental" nature of this approach is reflected (in whole or in part) in existing program analysis techniques aimed at fault localization (e.g., [11, 14, 20, 22]).

In this paper we show that a class of program analysis approaches exist that are completely experimental in nature. By this, we mean that these techniques can be characterized in terms of the guidelines and methodologies defined and practiced within the long-established paradigm of classical experimentation. Building on this characterization, we present an operational definition of a new paradigm for program analysis, that (following Zeller)[1] we call *experimental program analysis*, and we show how analysis techniques can be characterized in terms of this paradigm. We present three studies that show that an understanding of this paradigm can help researchers (1) identify limitations of analysis techniques, (2) improve existing experimental program analysis techniques, and (3) create new experimental program analysis techniques by adapting existing non-experimental techniques.

The remainder of this paper proceeds as follows. We begin by overviewing classical experimentation and relevant concepts (Section 2). Section 3 presents our descriptive and operational definitions of experimental program analysis, illustrates them by an example, and also describes several intriguing (and potentially exploitable) differences between experimental program analysis and classical experimentation. Section 4 presents our studies, which illustrate the potential benefits of using experimental program analysis. Section 5 describes related work, and Section 6 concludes.

2 Background: Experimentation

Classical experimentation is a mature field, and its methods are well-discussed in the literature (e.g., [3, 9, 12, 13, 15, 17, 18]). Since experimental program analysis techniques draw from classical experimentation by conducting experiments in order to conduct program analysis, in this section we distill, from that literature, an overview of the empirical method. In addition, we highlight the material about experiments that is most relevant to the understanding of experimental program analysis.

The initial step in any scientific endeavor in which a conjecture is meant to be tested using a set of collected observations is the **recognition and statement of the problem**. This activity involves formulating *research questions* that define the purpose and scope of the experiment, identifying the phenomena of interest, and possibly forming conjectures regarding the outcome of the questions. As part of this step, the investigator also identifies the target *population* to be studied, and on which conclusions will be drawn.

Depending on the outcome of this first step, as well as the conditions under which the investigation will take place,[2] different research strategies (e.g., case studies, surveys, experiments) may be employed to answer the formulated

[1] Although there are commonalities between our view of experimental program analysis and Zeller's, our background in experimentation leads us to several different views; Section 5 explains.

[2] Conditions may include the desired level of control over extraneous factors, the available resources, and the need for generalization.

research questions. These strategies have different features and involve different activities. In the case of experiments, the design strategy of interest in this paper, scientists seek to test hypothesized relationships between independent variables and dependent variables by manipulating the independent variables through a set of purposeful changes while carefully controlling extraneous conditions that might influence the dependent variable of interest. When considering experiments, the researcher must perform four distinct activities [12, 13]:

(1) Selection of independent and dependent variables. This activity involves the identification of the *factors* that might influence the outcome of the tests that will later be conducted on the identified population. The investigator must isolate the factors that will be manipulated (through purposeful changes) in investigating the population and testing the hypotheses; these isolated factors are referred to as *independent variables*. Other factors that are not manipulated, but whose effects are controlled for by ensuring that they do not change in a manner that could confound the effects of the independent variable's variations, are typically referred to as *fixed variables*. Variables whose effects cannot be completely controlled for, or variables that are simply not considered in the experiment design, are *nuisance variables*. The response or *dependent variables* measure the effect of the variations in the independent variables on the population. The observations elicited from the dependent variables are used in the experiment to conduct the tests that evaluate the predictions of the experiment.

(2) Choice of experiment design. Experiment design choice is concerned with structuring variables and data so that the predictions can be evaluated with as much power and as little cost as possible. The process begins with the researcher choosing, from the scales and ranges of the independent variables, specific levels of interest as *treatments* for the experiment. Next, the researcher formalizes the predictions about the potential effects of the treatments on the dependent variable through a *statement of hypotheses*. To reduce experimentation costs, the investigator must then determine how to *sample the population* while maintaining the generalization power of the experiment's findings. The investigator then decides how to *assign the selected treatments* to the units in the sample to efficiently maximize the power of the experiment, while controlling the fixed variables and reducing the potential impact of the nuisance variables, so that meaningful observations can be obtained. Finally, the investigator assesses the limitations of the experiment — formally known as *threats to the experiment's validity*. The types of threats that we consider in this paper are those identified by Trochim [16]: (1) threats to internal validity (could other factors affecting the dependent variables of the experiment be responsible for the results), (2) threats to construct validity (are the dependent variables of the experiment appropriate), (3) threats to external validity (to what extent could the results of the experiment be generalized), and (4) threats to conclusion validity (what are the limitations of the experiment's conclusions, and how could a stronger experiment be designed).

(3) Performing the experiment. This activity requires the codification and pursuit of specified *procedures* that properly gather observations in order to test the target hypotheses. These procedures should reduce the chance that the dependent variables will be affected by factors other than the independent variables, such as nuisance variables. Thus, the researcher must regularly monitor the implementation of the experiment procedures to reduce the chances of generating effects by such extraneous factors.

(4) Analysis and interpretation of data. An initial *data analysis* often includes the computation of measures of central tendency and dispersion that provide a quick characterization of the data, and that might help the researcher

identify anomalies worth revisiting. More formal data analysis includes statistical significance assessments on the effect of the treatments on the dependent variables. Such assessments provide measures of confidence in the reliability and validity of the results and help interpretation proceed in an objective and non-biased manner. The data analysis allows the researcher to *test the hypotheses* stated during the experiment's design in order to evaluate the effect of the treatments. The interpretation of the hypothesis testing activity during an *interim analysis* can lead to further hypothesis testing within the same experiment, either through the continued testing of current hypotheses or the formulation of new hypotheses. If more data is needed to test hypotheses, then the process returns to experiment design so that such data can be obtained; this establishes a "feedback loop" in which continued testing may take place during the course of the experiment. If more data is not needed, then the investigation proceeds to the final step in the process of experimentation.

The final step when performing any empirical study, regardless of the research strategy utilized, is the offering of **conclusions and recommendations.** A researcher summarizes the findings through *final conclusions* that are within the scope of the research questions and the limitations of the research strategy. However, studies are rarely performed in isolation, so these conclusions are often joined with recommendations that might include guidance toward the performance of replicated studies to validate or generalize the conclusions, or suggestions for the exploration of other conjectures.

3 Experimental Program Analysis

In this section we define and describe experimental program analysis, and then discuss several traits that distinguish it from classical experimentation and traditional program analysis. As a vehicle for this discussion, we illustrate the concepts that we present using an existing technique that (as we shall show) is aptly described as an "experimental program analysis" (EPA) technique — the technique implemented by HOWCOME [20].

HOWCOME is a tool intended to help engineers localize the cause of an observed program failure f in a failing execution e_f. HOWCOME attempts to isolate the minimal relevant variable value differences in program states that can reveal cause-effect chains describing why f occurred. To do this, HOWCOME applies a subset of e_f's variable values to the corresponding variables in a passing execution e_p, and tests whether the applied changes reproduce f.[3] If the applied subset "fails" the test (by not reproducing f), then a different subset is tested. If the subset "passes" the test (by reproducing f), then a subset of those incriminating variable values are tested. This process continues until no further subsets can be formed or are capable of reproducing the failure.

3.1 Definition and Illustration

We descriptively define experimental program analysis as follows:

> *Experimental program analysis* is the process of executing a series of evolving tests on a target program, or its artifacts or byproducts, under controlled conditions, in order to characterize or explain the effect of one or more independent variables on an aspect of the program.

[3] HOWCOME uses the Delta Debugging [22] algorithm, which is a more general form of experimental program analysis. To focus our discussion, in this paper we consider only the use of Delta Debugging in HOWCOME on variable values in program states.

One important characteristic of experimental program analysis is the notion of a series of "evolving tests". When performing classical experiments, tests are usually fixed in advance [13]. In the context of program analysis, however, multiple tests are often executed in sequence, with later tests leveraging the results of previous tests to more efficiently converge to a result (e.g., re-sampling the population and refining hypotheses). An experimental program analysis "process" is then necessary to manage the tests' evolution, including a feedback loop enabling previous results to guide which future tests will be conducted and under what conditions. In summary, the outcomes of current tests determine further tests to be conducted.

Second, the phrase "controlled conditions" relates to the notion of the purposeful changes that are unique to experiments. In the context of experimental program analysis, purposeful changes to the program, or program artifacts or byproducts of interest — for example, input, source code, descriptive properties, and environment properties — are intentionally chosen for application as treatments. Note that the application of these treatments is tightly associated with how the tests evolve, linking them to the "process" as well. "Controlled conditions" also refers, however, to the management and isolation of factors that may affect the dependent variables, thereby confounding the effect of the treatments and limiting what can be learned from the experiment.

Third, experimental program analysis is performed "in order to characterize or explain the effect of the independent variables on an aspect of the program". It is important to understand the nature of this characterization or explanation. In rare cases, experiments might be able to "prove" a conjecture if an entire population is observed. In practice, however, EPA techniques, like classical experiments, will operate on a sample of a population, leading to answers that are not absolutely certain, but rather highly probable or largely complete in scope. In general, then, the outcome of an EPA technique is a description or assessment of a population reflecting an aspect of the program of interest (e.g., "what is the potential behavior of the program?"), or the determination of likely relationships between the independent variable (treatments) and the dependent variable (e.g., "what inputs are making the program fail?").

To further elucidate this descriptive definition we augment it with an operational definition, presented in tabular form (Table 1). This table revisits the experimentation guidelines presented in Section 2 in light of the descriptive definition. The table also distinguishes between tasks performed by an EPA technique (gray rows), and tasks performed by the creator of the technique (white rows). Note that the tasks corresponding to the technique can be controlled and performed repeatedly (utilizing the experiment's feedback loop) by the process that guides the EPA technique.

Table 1 also summarizes how our example of the HOWCOME technique [20] relates to our operational definition. We discuss each task in the definition in detail, using HOWCOME to illustrate that task's role in experimental program analysis. We also discuss, for many tasks, the ways in which they can contribute to validity threats in the experiments conducted by EPA techniques.

3.1.1 Recognition and Statement of the Problem

This planning step is reflected in our operational definition by the "formulation of research questions" and "identification of population" tasks.

Formulation of research questions. This task guides and sets the scope for the experimental program analysis activity, focusing on particular aspects of a program.

Guideline	Task Identifier	Role in Experimental Program Analysis	HOWCOME
RECOGNITION AND STATEMENT OF THE PROBLEM	*Research questions*	Questions about specific aspects of a program.	"Given e_p and f in e_f, what are the minimum variable values in e_f that cause f?"
	Population	The aspect of the program that the experiment will draw conclusions about.	All program states $S_{e_p} \in e_p$.
SELECTION OF INDEPENDENT AND DEPENDENT VARIABLES	*Factors*	The internal or external aspects of the program that could impact the effect of the measured manipulations.	Variable value changes, failure-inducing circumstances, number of faults, outcome certainty, unchanging variable values, etc.
	Independent variables	The factors that are manipulated in order to impact the program aspect of interest.	Values of variables in e_f at each state $s \in S_{e_f}$.
	Fixed variables	The factors that are set or assumed to be constant.	Variable values that do not change.
	Nuisance variables	Uncontrolled factors that can affect the measured observations on the program.	Multiple failure-inducing circumstances, number of failures, outcome certainty, etc.
	Dependent variables	The constructs used to quantify the effect of treatments on the target program aspect.	Whether the execution reproduces f, succeeds as did e_p, or is an inconclusive result.
CHOICE OF EXPERIMENT DESIGN	*Treatments*	The specific instantiations of levels from the independent variables that are tested.	Difference in variable values between a state in e_p and the corresponding state in e_f.
	Hypothesis statement	Statements or conjectures about the effect of treatments on an aspect of the program.	A null hypothesis H_0 for each treatment is: "The value changes do not produce f in e_p."
	Sample	A set of elements from the population.	Program states in e_p.
	Treatment assignment	Assigns chosen levels of independent variables to the sampled units.	A compound treatment of variable values are applied to states in e_p.
	Threats to validity	Threats due to unexpected changes in the program or environment, unforeseen factors' dependencies, a biased sample, etc.	Cause-effect chain may not contain the true minimally relevant variables due to dependencies. Multiple faults may influence chain.
PERFORMING THE EXPERIMENT	*Experiment procedures*	An automated process using algorithms to assign treatments to units in the sample and capturing observations measuring the isolated effects of those changes.	An observation is collected for each test of variable value differences to a state.
ANALYSIS AND INTERPRETATION OF DATA	*Data analysis*	Analyzing the observations to discover or assess the effects of the treatments on the aspect of the program of interest.	The effects of applying the variable values is gauged by observing the effect on the execution's output.
	Hypothesis testing	Using the analysis from the observations to confirm or reject the previously-stated hypotheses regarding treatment effects.	H_0 is rejected if a treatment reproduces f, not rejected if the execution passes, and not rejected if the outcome is inconclusive.
	Interim analysis	Making a decision regarding whether further tests are needed based on the results of the current and previous tests.	If H_0 was rejected and subsets of variable values can be formed, design new tests on those values. Otherwise choose different values from those remaining (if any remain).
CONCLUSIONS AND RECOMMENDATIONS	*Final conclusions*	The conclusions drawn from the application of experimental program analysis.	Using the reported cause-effect chain, or deciding to generate a new chain.

Table 1: Key tasks in experimental program analysis. Each task is summarized in the third column. The tasks are grouped, in the first column, according to the experimentation guidelines in Section 2. The fourth column uses HOWCOME to illustrate each task. Tasks performed by experimental program analysis are in gray rows, while tasks in white rows are performed by investigators.

> *Example:* in HOWCOME, the research question is: "given a passing execution e_p and failure f in a failing execution e_f, what are the minimum variables $V \in e_f$ that cause f?"

Identification of population. This task identifies the aspect of the program about which inferences will be made. Although experimental program analysis studies software systems, the population universe of its experiments is generally some set of artifacts or byproducts of interest of a program.

> *Example:* in HOWCOME, conclusions about variable values relevant to f are made by applying those values to program states in e_p. Each state is a unit in the population of all program states S_{e_p}.

3.1.2 Selection of Independent and Dependent Vars

The outcomes of this step are the identification of (1) the aspect of the program that will be manipulated by the EPA technique, (2) a construct that will be used to measure the effects of these manipulations on the program, and (3) the factors for which the experiment performed does not account that could bias observations.

Identification of factors. This task identifies any internal or external aspect of the program and environment that could impact the effect of the purposeful changes that are being measured. An important byproduct of this step is the awareness of potentially confounding effects on the results.

> *Example:* in HOWCOME, factors include: variable values — both those that are changed in a single test and those that are not — that can impact the final execution output; number of faults, as multiple faults may induce different failure circumstances; the failure-inducing circumstances themselves, including non-deterministic or synchronization issues on which failures may depend; and potential uncertainty about outcomes (oracle problems).

Selection of independent variables. This task identifies the factors that will be manipulated throughout the experiment performed by the EPA technique in order to support the analysis of the program under investigation. As in classical experiments, treatments are selected as levels from the ranges of the independent variables.

> *Example:* in HOWCOME, the independent variable is the values of the variables in e_f at each program state. (The operative notion is that through modification of this variable, HOWCOME may find different variable values to be relevant to f). As an example of treatment design, if the variable x is an 8-bit unsigned integer, then the range of the independent variable is 0–255, and a treatment from x is one of the 256 possible values (i.e., levels) of x.

Selection of fixed variables. This task chooses a subset of factors to hold at fixed levels. Making properties constant reduces the likelihood that they will affect the dependent variable in unexpected or confounding ways. This is one way in which EPA techniques, like classical experiments, can reduce threats to validity: by ensuring that extraneous factors do not impact results.

> *Example:* in HOWCOME, the variable values that are not manipulated between e_p and e_f are kept constant to control their effect on the program outcome. This attempts to prevent any variable values other than those in the treatment being tested from influencing the execution's output.

Identification of nuisance variables. This task identifies uncontrolled variables. These factors may intentionally be left uncontrolled because it may not be cost-effective to control them, or because it is not possible to do so. In any case, it is important to acknowledge their presence so that an EPA technique's conclusions can be considered in light of the possible role of these factors, which are threats to the internal validity of the technique's experiments. (As we shall see in Section 4.1, improvements to EPA techniques can come in the form of finding ways to reduce, or even eliminate, the impact of these nuisance variables.)

Example: in HOWCOME, the presence of multiple faults and the existence of multiple failure-inducing scenarios, or scenarios for which the outcome is not certain and cannot be classified as passing or failing, are nuisance variables.

Selection of dependent variable(s). This task determines how the effects of the purposeful changes to the independent variable (treatments) will be measured. A construct is then chosen that captures these measurements in the form of observations that can be analyzed to test the treatments. If this task is not performed properly, then the construct validity of the technique may be threatened, as the construct may not capture the effect that it is intended to.

Example: in HOWCOME, the dependent variable involves whether f is reproduced, and the construct is a testing function that indicates whether (1) the execution succeeded as did the original, unmodified execution e_p; (2) f was reproduced by the treatments; or (3) the execution's output was inconclusive, as when inconsistent program states occur due to the modification of certain variables.

3.1.3 Choice of Experiment Design

The choice of experiment design is a crucial step for maximizing the control of the sources of variation in the program and the environment, and reducing the cost of an experimental program analysis technique. Tasks in the choice of experiment design begin those that can be controlled by the process driving the EPA technique; this includes the opportunity to leverage the results from previous tests to influence how the tasks are performed.

Design of treatments. This task determines specific levels from each independent variable's range at which to instantiate treatments. If there are multiple independent variables, or if multiple levels from the same variable are to be combined, then this task also determines how the instantiations will be grouped together to form compound treatments. It is these treatments that are tested in experimental program analysis, using units from the population, and about which conclusions will be drawn.

Example: in HOWCOME, tests are crafted through the selection of potential variable values — levels of the independent variable — to apply to a program state in e_p. Each variable change is an instantiation of the difference between the variable value in e_p and the corresponding value in e_f. (These value changes are tested to see if f is reproduced. If so, then either the variable value changes will be used to create a cause-effect chain or an attempt will be made to narrow those values further.) Thus, potentially relevant variable value changes (to a program state in e_p) consist of a compound treatment in HOWCOME's tests.

Formulate hypotheses. This task involves formalizing the conjectures about the effects of treatments on the aspect of the program of interest. These hypotheses are later evaluated after observations are collected about the treatments' effects so that experimental program analysis can draw conclusions about the treatments' impact on the population of interest.

Example: in HOWCOME, when considering potential variable value changes to a program state in e_p, tests assess whether the variable values reproduce f in the execution. A null hypothesis for a particular

treatment of variable value differences therefore states that "the variable value changes do not reproduce f in e_p."[4]

Sample the population. A sample is a set of elements from the population. Sampling the population by collecting observations on a subset of the population is one way by which experimental program analysis achieves results while incurring acceptable costs. This step defines the sampling process (e.g., randomized, convenience, adaptive) and a stopping rule (e.g., execution time, confidence in inferences) to apply to the analysis. If this task is not completed properly, the external validity of the experiment may be threatened, as conclusions may not generalize to the population.

Example: in HOWCOME, program states from e_p are sampled so that variable values from equivalent states in e_f can be applied. States at which relevant, minimal variables are found are used to form the reported cause-effect chain.

Assign treatments to sample. This task involves assigning treatments to one or more experimental units in the sample. Some assignment possibilities include random assignment, blocking the units of the sample into groups to ensure that treatments are better distributed, and assigning more than one treatment to each experimental unit. The result is a set of objects of analysis from which observations will be obtained to test the treatments during experimental program analysis. If this task is not performed correctly, the experiment could suffer from conclusion validity problems, as its conclusions may not be powerful enough due to issues such as having insufficient replications of tests for each treatment.

Example: in HOWCOME, the variable value differences selected as treatments are applied to a program state in e_p so that it can be observed whether the value changes reproduce f.

Identify threats to validity. This task identifies the threats to the EPA technique's validity such as unexpected changes in the program or environment, unforeseen dependencies, and biased sampling. Understanding validity threats enables a more objective interpretation of the results in light of the technique's weaknesses.

Example: in HOWCOME, the cause-effect chain may not contain the true minimally relevant variables due to the particular executions e_p and e_f used. Also, multiple faults and various failure-inducing circumstances may influence the cause-effect chain.

3.1.4 Performing the Experiment

This step is primarily mechanical and consists of just one task: obtaining a set of observations on the sampled units, according to experimental procedures, that measure the effect of the independent variable manipulations (treatments). If this task is not performed correctly, the experiment's internal validity may be affected due to extraneous factors influencing the observations obtained.

Execute experimental procedures. In experimental program analysis, this procedure can be represented algorithmically and can be automated, which contrasts it with experiments in other sciences where the process of following

[4] Although null hypotheses are used here as a vehicle to analyze the impact of treatment variable value differences, hypotheses regarding treatments could be stated in many forms — as long as their evaluation leads to meaningful analysis regarding the effects of treatments and the purposeful changes to be performed next.

experimental procedures and collecting observations is performed by researchers. The observations obtained during this task are those that relate to the dependent variable, and should capture only the isolated effects on that variable that follow from the application of the treatment.

> *Example:* in HOWCOME, an observation is collected for each test of variable value changes to a state. The process of conducting a test involves running e_p, interrupting execution at the program state s_i under consideration, applying the treatment variable values from e_f to e_p at s_i, resuming the execution of e_p, and determining whether (1) e_p succeeded, (2) f was reproduced, or (3) the execution terminated with an inconclusive result. The outcome of this test is an observation collected by the experiment.

3.1.5 Analysis and Interpretation of Data

Experimental techniques must analyze observations to evaluate hypotheses and determine the next course of action. To ensure that conclusions are objective when a population sample has been used, statistical measures can assign a level of confidence in the results, gauging the effectiveness and efficiency of the experiment. This task is generally automated by EPA techniques. If this task is not performed properly, the conclusion validity of the experiment can be threatened due to the use of a statistical test of insufficient power.

Performing data analysis. This task involves analysis of collected observations for the purpose of evaluating the previously-stated hypotheses. In many cases, this may involve statistical analyses to determine the appropriateness of the decision made regarding an hypothesis. (In Section 4 we consider a technique that does this.)

> *Example:* in HOWCOME, the only information needed to evaluate hypotheses is whether or not the dependent variable indicated that the variable value treatment caused f to be reproduced, or whether it caused an inconclusive result.

Testing of hypotheses. Hypothesis testing is used to assess the effect of the purposeful changes made by the investigator (or, in the case of experimental program analysis, by the automated process that manages the experiment).

> *Example:* in HOWCOME, the null hypothesis H_0 is rejected if the variable value treatment reproduces the original failure. This tells the technique that it should concentrate on trying to find the minimally relevant variable value differences within this treatment. As such, the rejection of H_0 guides the purposeful changes in the independent variable (i.e., guides the design of future treatments) by determining whether the particular treatment variables should be refined during the remainder of experimental program analysis.

Performing interim analysis. After the hypotheses have been tested, a decision must be made about whether further treatments need to be tested or different experimental conditions need to be explored. EPA techniques determine automatically, based on the results of previous tests, whether to continue "looping" (i.e., further purposeful changes will be made to the independent variables), or whether the experimental program analysis has reached a point where the technique can conclude and output results.

Example: in HOWCOME, if H_0 (for the treatment variable value differences) were rejected, indicating that those treatments reproduced f, then new tests will be designed from those values to further minimize the variable values relevant to f (if further minimizations are possible). Otherwise, if different sets of variable values remain that have not yet been tested, they will be tested as treatments next. When no variable values remain to be tested, the cause-effect chain is reported by combining the isolated variable value differences into a sequential report explaining the causes of f.

3.1.6 Conclusions and Recommendations

Investigators must draw conclusions based on experimental program analysis results and, if appropriate, recommend future courses of action, which can include the use of the EPA technique's output or a repeated or refined run of the technique.

Draw final conclusions. These are the final conclusions drawn from experimental program analysis when the analysis is complete and no further testing is needed or can be done.

Example: in HOWCOME, a cause-effect chain can be used by engineers to track the root cause of the failure through its intermediate effects and ultimately to the failure itself, or to select different passing and failing executions to provide to HOWCOME.

3.2 Experimental Program Analysis Versus Classical Experimentation

Experimental program analysis has traits that distinguish it from classical experimentation, and also has several implications for program analysis; we now comment on several of these.

Replicability and sources of variation. Program analysis activities are not subject to certain sources of spurious variations that are common in other fields. For example, program analysis is usually automated, reducing sources of variation introduced by humans, which are among the most difficult to control and measure reliably. We have also observed that some typical threats to replicability must be reinterpreted in the context of experimental program analysis. For example, the concept of learning effects (where the behavior of a unit of analysis is affected by the application of repeated treatments) should be reinterpreted in the program analysis context as residual effects caused by incomplete setup and cleanup procedures (e.g., a test outcome depends on the results of previous tests). Also, a software system being monitored may be affected by the instrumentation that enables monitoring, and this resembles the concept of "testing effects" seen in classical experimentation.

Experimental program analysis *is* susceptible to sources of variation that may not be cost-effective to control. For example, non-deterministic system behavior may introduce inconsistencies that lead to inaccurate inferences. Controlling for such behavior (e.g., manipulating the scheduler) may threaten the generality of an EPA technique's findings. Still, experimental program analysis has the advantage of dealing with software systems, which are abstractions and are more easily controlled than natural occurring phenomenon.

The cost of applying treatments. In most cases, software systems have relatively low execution costs — especially in comparison, say, to the cost of inducing a genetic disorder in a population of mice, and then applying a treatment to

this population. Systems may thus be exercised many times during the software development and validation process. This is advantageous for experimental program analysis because it implies that multiple treatment applications, and multiple hypothesis tests, are affordable. Two factors contribute to this trait. First, in experimental program analysis, application of treatments to experimental units is often automated and requires limited human intervention. Second, there are no truly expendable units or treatments; that is, the aspects of the system that are being manipulated or the sampled inputs can be reused without incurring additional costs (except for the default operational costs). EPA techniques can take advantage of this trait by using increased sample sizes to increase the confidence in, or quality of, the findings, and adding additional treatments to their design in order to learn more about the research questions.

Sampling the input space. Experiments often sample from a population in order to draw inferences from the sample that reflect the population. The power of EPA techniques to generalize and the correctness of their inferences will be dependent on the quality of the samples that they use. Although this challenge is not exclusive to experimental program analysis (e.g., software testing attempts to select "worthwhile" inputs to drive a system's execution) and there will always be uncertainty when making inductive inferences, we expect the uncertainty of EPA techniques to be measurable by statistical methods if the sample has been properly drawn and the assumptions of the method have been met.

Assumptions about populations. Software systems are not naturally occurring phenomena with distributions that follow commonly observed patterns. Experimental program analysis data reflecting program behavior is, for example, rarely normal, uniform, or made up of independent observations. This limits the opportunity for the application of the statistical analysis techniques most commonly used in classical experimentation. One alternative is to apply data transformations to obtain "regular" distributions and enable traditional analyses. However, existing transformations may be unsuited to handling the heterogeneity and variability of data in this domain. Instead, it may be necessary to explore the use of "robust" analysis methods — that is, methods that are the least dependent on the failure of certain assumptions.

Procedures versus algorithms. EPA techniques are unique in at least two procedural aspects. First, whereas procedures in classical experimentation are typically lists of steps to be taken by humans, procedures in experimental program analysis are commonly represented as algorithms. The algorithmic representation naturally lends itself to both analysis and automation, which reduces its application costs. Second, these algorithms can be extended to manage multiple experimental tasks (classical experimental procedures focus just on the execution). For example, HOWCOME utilizes an algorithm to refine the stated hypothesis within the same experiment and guide successive treatment applications of variable values to a program state. We strongly suspect that other tasks in experimental program analysis, such as the choice of experimental design and sampling procedures, can be represented in algorithmic form.

The role of feedback. Classical experimentation typically advocates the separation of data collection from data analysis activities. In one class of classical experimentation approaches — sequential analysis — on the other hand, data is analyzed as it is collected to establish a feedback loop that can drive the sampling process or enable an early stopping of the experimental process [15]. This second type of analysis is common, for example, in clinical trials where the experiment stops if the superiority of a treatment is clearly established, or if adverse side effects become obvious, resulting in less data collection, which reduces the overall costs of the experiment. EPA techniques can take

sequential analysis with feedback even further, interleaving not only the sampling process and the stopping rules, but also determining what design to use and what treatments to apply based on the data collected so far. This will enable EPA techniques to scale in the presence of programs with large populations or a large number of potential treatments, whereas the application of classical experimental approach would not be affordable in such cases.

4 Applications of the Experimental Program Analysis Paradigm

We now consider in detail the three applications of the experimental program analysis paradigm described in Section 1.

4.1 Assessing EPA Techniques

Our operational definition of experimental program analysis provides a "framework" for assessing the limitations of existing EPA techniques. By mapping a technique to the framework, technique limitations can be assessed in two ways. First, our operational definition specifically outlines each high-level task in techniques' experiments so that incorrectly approached tasks, or tasks staged to rely on assumptions that may not be met, can be more easily identified. For example, experimental design errors, sampling bias, confusion of correlation and causality, and interactions between effects can all intrude on EPA techniques and bias or limit their results. (This is especially likely if the researchers creating techniques are unaware of the possibility of such limitations.) Second, our operational definition explicitly considers validity threats that are inherent in controlled experimentation. Although considering the limitations of analysis techniques is not new to the program analysis community, explicitly considering the validity threats that typically accompany the design or assessment of experiments provides a new lens through which to view such limitations in EPA techniques.

Assessing techniques' limitations is important, of course, so that conclusions can be appropriately interpreted; however, after assessing limitations, improvements that mitigate or overcome such limitations can naturally suggest themselves. Thus, our example of assessing an EPA technique also includes such improvements.

Detailed example of assessing techniques. In Section 3.1 and Table 1, we outlined some of the validity threats to the HOWCOME technique. For example, HOWCOME will isolate the minimally relevant variable values between the provided passing and failing execution. However, these may not be the *truly minimal* variables and values causing the observed failure: rather, they may just be the minimal variable values given the subsets isolated by the divide-and-conquer algorithm for the given executions.[5] The possibility of dependencies between variables influencing those isolated as minimally relevant in the cause-effect chain is another threat not explicitly outlined in [20]. Although such dependencies may be part of the causality chain between values that ultimately results in the failure, understanding the impact of these nuisance variables could help in isolating the fault, or in doing so more quickly (e.g., by manipulating the values of the variables while keeping the values of those on which they are dependent constant).

Last, not all of the approach's scalability limitations have been specifically identified in [20]. Although systematic, HOWCOME's sampling procedures and strategies for selecting new variable values to test may become impractical

[5] In earlier work pertaining to input minimization [22], this threat is discussed in terms of "global minimums" versus "local minimums". Local minimums are detected by HOWCOME, whereas a "global minimum" could be detected by exhaustively testing all combination of variable-value changes to program states.

(slow to converge) in the presence of programs with many variables and large traces. It could be valuable, for example, to consider adaptive sampling mechanisms which trade some of the technique's accuracy for scalability or efficiency. Another possibility would be to utilize random sampling over the program state space; the size of the sample could be based on the availability of resources (e.g., time). Although the use of random sampling could result in missing information in the cause-effect chain, it would provide an opportunity to employ statistical analysis to generate confidence as to the accuracy (and perhaps completeness) of the cause-effect chain thus created.

Because an implementation of HOWCOME was not available to us, we do not explore such improvements further in this paper; instead, in the upcoming discussion we focus our data collection efforts on another technique.

4.2 Creating New EPA Techniques

Our operational definition of experimental program analysis provides a "recipe" for creating new EPA techniques. Creating a new EPA technique can follow either of two approaches. The first approach involves modifying an existing program analysis technique so that it conducts experimental program analysis. Although this does not create a new program analysis technique per se, it does create a new *experimental* program analysis technique, and so we group it into this application. The second approach involves designing a completely new technique to approach a program analysis problem in a manner that has not been done before. We now provide an example of the first approach.

Detailed example of creating new techniques. Daikon [7] is an implementation of a program analysis technique that infers likely program invariants from execution traces using a "library" of predefined invariant types. At each program point of interest, all possible invariants that might be true are tested by observing the values of variables, during program executions, at the program points of interest. If an invariant is violated in an execution, it is discarded (falsified). If an invariant has not been falsified in any execution and has been tested enough that Daikon has sufficient (statistical) confidence in its validity, it is reported as a likely invariant.

As viewed in relation to our experimental program analysis operational definition, Daikon tests relationships about variables (in the form of invariants); the population of possible relationships is what an investigator wishes to learn about. Since there are many possible relationships with which one could describe variables, and it is not feasible to test all of them, Daikon chooses a "sample" of these relationships (as per Daikon's predefined invariant types) for testing using program executions.

As presented in [7], however, Daikon is not an EPA technique because it does not conduct a series of *evolving* tests (with purposeful changes) in order to investigate variable relationships. If Daikon were conducting experimental program analysis, it would likely manipulate the executions that are provided to test the invariants; however, Daikon instead simply tests all variable values in all execution traces against all applicable invariants, without concern for the manner in which those traces are used to test invariants.[6] One reason to consider adding such manipulations to Daikon and transforming it into an EPA technique is for possible improvement, such as increasing the technique's cost-effectiveness.

[6] An alternative mapping of Daikon to the EA framework could consider the potential program behavior as the population, the traces as the sample, and the invariant and program points as the independent variable. However, such a mapping is inadequate because Daikon's invariants are not manipulated but rather provided in advance, and the notion of testing an invariant cannot be perceived as applying a treatment to a program. In general, we have found, and the Daikon case illustrates, that the act of mapping techniques to the "framework" provided by our operational definition is an iterative activity that provides insights into the technique as the mapping converges.

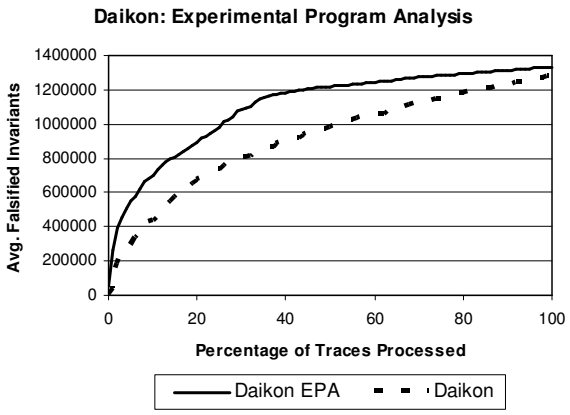

Figure 1: The average rate of invariant falsification for the Daikon$_{EPA}$ (solid line) and Daikon (dashed line) techniques.

One way in which Daikon *could* manipulate execution traces is to manipulate the *order* in which they are used to test possible invariants. This notion leads to the idea of choosing the next trace to be processed based on each trace's likelihood of falsifying candidate invariants. (Falsifying invariants quickly helps reduce the effort spent in testing invariants that will eventually be falsified later.)

Clearly, there are many ways to estimate how likely it is that an execution trace will falsify many invariants. We choose an approach based on each trace's coverage of program points. Each time a trace needs to be analyzed against all candidate invariants, we choose the trace t that covers the program point p with the most invariants remaining. When more than one trace covers p, we use the total number of program points (at which candidate invariants remain) that are covered by the traces as the tie-breaker. We term this new technique Daikon$_{EPA}$. A null hypothesis H_0 can be stated, "p will no longer have the most remaining candidate invariants after t is processed." H_0 is rejected if t falsifies enough invariants such that p no longer has the most invariants, and the next trace is selected based on a new program point. Otherwise, p is again used to select the next trace.

To investigate whether Daikon$_{EPA}$ falsifies invariants more efficiently than the original Daikon [7], we implemented our execution trace manipulation scheme in Daikon version 3.1.7. We then conducted a case study to investigate the capabilities of Daikon$_{EPA}$, using the `Space` software system as a subject. `Space` is written C, and contains 136 functions and 6,218 lines of non-commented, non-blank source code. This system, along with numerous test suites, is available as part of an infrastructure supporting experimentation [4].

We randomly selected five branch-coverage-adequate test suites for `Space`, and generated trace files for each execution in each test suite. We then used Daikon and Daikon$_{EPA}$ to detect invariants using all five suites. As a construct to compare the techniques, we measured the rate at which invariants were falsified.

Figure 1 compares the rate of invariant falsification of Daikon$_{EPA}$ and Daikon. The average number of invariants falsified (y-axis) is plotted against the percentage of the test suite that has been processed (x-axis). As the figure shows, by manipulating the order of the execution traces based on the results of its hypothesis testing, Daikon$_{EPA}$ falsified invariants more rapidly than did the original, non-experimental Daikon technique — especially at the beginning of the test suite — and at the expense of very little overhead. In fact, in our five test suites, Daikon$_{EPA}$ considered fewer invariants at all stages of processing execution traces; even after processing just one trace file, Daikon$_{EPA}$ falsified

from *six to 37 times* more invariants than Daikon. As a result, this new approach could be particularly advantageous in situations in which invariant detection time is limited, and may be prematurely terminated.

4.3 Improving Existing EPA Techniques

We expect that our operational definition will expose many opportunities to utilize established experiment design strategies to improve EPA techniques. As one example, there are many strategies for assigning treatments to the units of a sample (e.g., block designs, factorial designs, split-plot designs), and these designs could provide opportunities to increase the power of experiments and lower experiment costs (e.g., a block design can isolate nuisance variables into blocks, removing them as error effects, and a split-plot design can reduce the need for applying combined treatments). Through the experimental program analysis paradigm we explicitly expose the program analysis community to such opportunities.

Detailed example of improving existing techniques. Sequential analysis [15] is an "incremental" form of hypothesis testing in which the effects of a treatment or sequence of treatments are evaluated throughout the experiment — not just at the end of a batch of tests. Each evaluation point can have three possible outcomes: (1) there is enough evidence to accept hypotheses, (2) the results are inconclusive and more tests are needed, or (3) there is enough evidence to reject hypotheses.

From a sequential analysis perspective, an opportunity exists for Daikon$_{EPA}$ to *immediately* consider whether there is enough evidence to conclude that a likely invariant will hold without considering the entire sample (analogous to the third sequential analysis outcome just listed). Daikon uses a confidence estimation process for all invariants that have not yet been falsified after all executions have been considered. Incorporating sequential analysis into Daikon involves adapting the technique's confidence estimation process (along the lines of sequential probability ratio tests [17]) and executing it at the end of *each* observation or *a group* of observations. (By an "observation", we refer to a single comparison of variable values in an execution trace to a relevant invariant.) The advantage of using sequential analysis is that reporting invariants as soon as their desired confidence levels are achieved may save many unnecessary observations that "further validate" likely invariants. However, sequential analysis may also increase the number invariants reported that are not really true (false positives), as some of these "early-reported invariants" may have been later falsified had they not been reported when the desired confidence level was met.

We implemented sequential analysis into Daikon$_{EPA}$; we term this new EPA technique Daikon$_{EPA.sa}$. In Daikon$_{EPA.sa}$, the confidence of each invariant's accuracy is tested at the end of each trace file (although this could be done at other places, such as after processing each observation). To assess Daikon$_{EPA.sa}$, we conducted a second case study on the same five `Space` test suites as in Section 4.2. In this study, we trace the number of observations saved through the use of sequential analysis. We also tracked the number of early false positives reported in each test suite using Daikon$_{EPA.sa}$ with a confidence limit of 99.99% (i.e., invariants are reported as soon as statistical tests indicate a 99.99% confidence in each invariant's validity).

Figure 2 depicts the performance of Daikon$_{es.sa}$ in terms of the observations (left y-axis, solid lines) and the invariants (right y-axis, dashed lines), as functions of the number of traces considered. The number of false positives reported appears to level off after approximately 25% of the traces have been processed. In the end, on average 16% of the invariants that were reported early by Daikon$_{EPA.sa}$ were false positives. On a more positive note, the solid

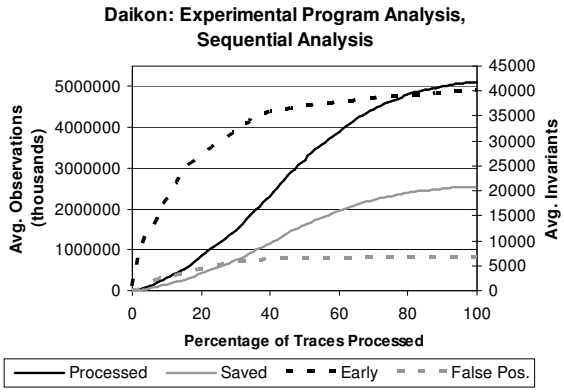

Figure 2: The average number of observations processed and saved, and the early and false-positive invariants reported by Daikon$_{EPA.sa}$.

curves for the number of observations indicate that the gains achieved through sequential analysis accumulate as more traces are analyzed. In the end, an average of *50%*, or over 2.54 billion, of the observations considered would be saved through sequential analysis. Thus, if performance savings are important, which may be the case for large software systems for which many execution traces must be considered, Daikon$_{EPA.sa}$ may be a more practical technique to ensure that *some* useful results can be reported, even if the precision of those results suffers.

5 Related Work

There is a growing body of knowledge on the employment of experimentation to assess the performance of, and evaluate hypotheses related to, software engineering methodologies, techniques, and tools. For example, Wohlin et al. [18] introduce an experimental process tailored to the software engineering domain, Fenton and Pfleeger [8] describe the application of measurement theory in software engineering experimentation, Basili et al. [2] illustrate how to build software engineering knowledge through a family of experiments, and Kitchenham et al. [10] provide guidelines for conducting empirical studies in software engineering.

There are also instances in which software engineering techniques utilize experimental principles as part of their operation (not just for hypothesis testing). For example, the concept of sampling is broadly used in software profiling techniques to reduce their associated overhead [1, 11], and experimental designs are utilized in interaction testing to drive an economic selection of combinations of components to achieve a target coverage level (e.g., [5]).

Within the program analysis domain, to the best of our knowledge, Zeller is the first to have used the term "experimental" in application to a program analysis techniques [21]. Our work differs from Zeller's, however, in two important ways.

First, Zeller's goal was not to precisely define experimental program analysis, but rather to provide a "rough classification" of program analysis approaches and "to show their common benefits and limits", and in so doing, to challenge researchers to overcome those limits [21, page 1]. Thus, in discussing specific analysis approaches, Zeller provides only informal definitions. In this work, we accept Zeller's challenge and apply our understanding of and

experience with controlled experimentation to provide a more precise notion of what experimental program analysis is and can be.

Second, our view of experimental program analysis differs from Zeller's in several ways. He writes that: "Experimental program analysis generates findings from multiple executions of the program, where the executions are controlled by the tool", and he suggests that such approaches involve attempts to "prove actual causality", through an (automated) series of experiments that refine and reject hypotheses [21, page 3]. When considering the rich literature on classical experimentation, there are several drawbacks in the foregoing suggestions. Experimentation in the scientific arena can be exploratory, descriptive, and explanatory, attempting not just to establish causality but, more broadly, to establish relationships and characterize a population [9, 12, 13]. For example, a non-causal question that can clearly be addressed by experimentation is, "is the effect of drug A applied to a subject afflicted by disease D more beneficial than the effect of drug B?" EPA techniques can act similarly — for example, with our improvements, Daikon$_{EPA}$ attempts to characterize program behavior, not establish causal relationships, and yet it is clearly experimental. Further, experimentation (except in a few situations) does not provide "proofs"; rather, it provides probabilistic answers — e.g., in the form of statistical correlations.

Finally, Zeller's explication contains no discussion of several concepts that are integral to experimentation, including the roles of population and sample selection, identification of relevant factors, selection of dependent and independent variables and treatments, experiment design, and statistical analysis. He also does not discuss in detail the nature of "control", which requires careful consideration of nuisance variables and various forms of threats to external, internal, construct, and conclusion validity. All of these quintessentially experimentation-related notions are present in our definition, and the utility of including them is supported.

One additional question of interest involves the relationship between experimental program analysis and other "types" of analyses, such as "static" and "dynamic" analysis. The characteristics of and relationships between techniques, and taxonomies of techniques, have been a topic of interest in many research papers (see, e.g., [6, 19, 21]. Our goal in this paper is not to taxonomize; nevertheless, we would suggest that experimental program analysis is not constrained to the traditional static or dynamic classification, but rather, is orthogonal to it. The experimental program analysis paradigm focuses on the *type* of analysis performed: namely, whether tests and purposeful changes are used to analyze software systems. As such, experimental analysis fills a gap that is not addressed by static or dynamic analysis techniques by offering (1) procedures for systematically controlling sources of variation, (2) experimental designs and sampling techniques to reduce the costs of experimentation, and (3) mechanisms to generate confidence measures in the reliability and validity of the results.

6 Conclusions

This paper has presented experimental program analysis as a new program analysis paradigm. We have shown that by following this paradigm, and using our operational definition of experimental analysis, it is possible to identify limitations of EPA techniques, improve existing EPA techniques, and create new EPA techniques by adapting existing non-experimental techniques.

There are many intriguing avenues for future work on experimental program analysis. One direction involves the use of the paradigm to solve software engineering problems in more cost-effective ways by adapting existing non-

experimental techniques or creating new EPA techniques. In this paper we have considered only a few examples of how to adapt existing techniques, but there are many others. For example, consider testing techniques whose goal is to select input values that expose faults. These techniques are primarily sampling strategies with some levels of control, but they lack the evolving tests and manipulation of the independent variable that would allow them to be considered EPA techniques. The incorporation of such elements could result in testing techniques that, based on previous results from exercising certain inputs, could better target the fault prone areas (just as Zeller has done when creating Delta Debugging). We conjecture that investigating such challenges from an experimental program analysis perspective can reveal new opportunities on how to address such software engineering challenges.

A second direction for future work, as we have mentioned, involves the automation opportunities for EPA techniques. Thus far, we have focused on the automation of experimental program analysis tasks and the advantages therein, but little else. However, it seems likely that the selection of the *approach* for a task can be automated as well. For example, EPA techniques could be encoded to consider multiple experimental designs (e.g., blocking, factorial, split-plot, latin square), and select that which is best suited for a specific instance of a problem. Improvements such as these may allow techniques to perform more efficiently, thereby making them more affordable to solve different classes of problems.

A third direction for future work with somewhat broader potential impacts involves recognizing and exploiting differences between experimental program analysis and classical experimentation. As Section 3.2 points out, there are several such interesting differences including, for example, the potential for EPA techniques to cost-effectively consider enormous numbers of treatments. It is likely that further study of experimental program analysis will open up intriguing new problems in the fields of empirical science and statistical analysis.

In closing, we believe that experimental program analysis provides numerous opportunities for program analysis and software engineering research. We believe that it offers distinct advantages over other forms of analysis — at least for particular classes of analysis tasks — including procedures for systematically controlling sources of variation in order to experimentally analyze software systems, and experimental designs and sampling techniques that reduce the cost of generalizing targeted aspects of a program. We believe that such advantages will lead to significant advances in program analysis research and in the associated software engineering technologies that this research intends to improve.

Acknowledgments

We thank Kent Eskridge and David Marx of the Statistics Department at the University of Nebraska–Lincoln for feedback on our definition of experimental program analysis. This work has been supported by NSF under awards CNS-0454203, CCF-0440452, and CCR-0347518 to University of Nebraska–Lincoln. We also thank the anonymous reviewers of an earlier version of this paper for comments that substantially improved the content of the work.

References

[1] M. Arnold and B. G. Ryder. A framework for reducing the cost of instrumented code. In *Proc. ACM SIGPLAN 2001 Conf. Prog. Lang. Design and Implementation*, pages 168–179, Snowbird, UT, USA, Jun. 2001.

[2] V. R. Basili, F. Shull, and F. Lanubile. Using experiments to build a body of knowledge. In *NASA Softw. Eng. Wshop.*, pages 265–282, Dec. 1999.

[3] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters: An Introduction to Design, Data Analysis, and Model Building*. Series in Probability and Mathematical Statistics. Wiley, New York, NY, USA, 1^{st} edition, 1978.

[4] H. Do, S. Elbaum, and G. Rothermel. Infrastructure support for controlled experimentation with software testing and regression testing techniques. In *Proc. Int'l. Symp. Empirical Softw. Eng.*, pages 60–70, Aug. 2004.

[5] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino. Applying design of experiments to software testing: Experience report. In *Proc.* 19^{th} *Int'l. Conf. Softw. Eng.*, pages 205–215, Boston, MA, USA, May 1997.

[6] M. D. Ernst. Static and dynamic analysis: Synergy and duality. In *Proc. ICSE'03 Wshop. Dynamic Anal.*, pages 24–27, Portland, OR, USA, May 2003.

[7] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Trans. Softw. Eng.*, 27(2):99–123, Feb. 2001.

[8] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. Course Technology, 2^{nd} edition, 1998.

[9] R. E. Kirk. *Experimental Design: Procedures for the Behavioral Sciences*. Brooks/Cole Publishing Company, 3^{rd} edition, 1995.

[10] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.*, 28(8):721–734, Aug. 2002.

[11] B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan. Scalable statistical bug isolation. In *Proc. ACM SIGPLAN 2005 Conf. Prog. Lang. Design and Implementation*, pages 15–26, Chicago, IL, USA, Jun. 2005.

[12] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, USA, 4^{th} edition, 1997.

[13] C. Robson. *Real World Research*. Blackwell Publishers, Inc., Malden, MA, USA, 2^{nd} edition, 2002.

[14] J. R. Ruthruff, M. Burnett, and G. Rothermel. An empirical study of fault localization for end-user programmers. In *Proc.* 27^{th} *Int'l. Conf. Softw. Eng.*, pages 352–361, May 2005.

[15] D. Siegmund. *Sequential Analysis: Tests and Confidence Intervals*. Springer-Verlag, New York, NY, USA, 1985.

[16] W. M. K. Trochim. *The Research Methods Knowledge Base*. Atomic Dog Publishing, Cincinnati, OH, USA, 2^{nd} edition, 2001.

[17] A. Wald. *Sequential Analysis*. Wiley, New York, NY, USA, 1947.

[18] C. Wohlin, P. Runeson, M. Host, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Kluwer Academic Publishers, Boston, MA, USA, 2000.

[19] M. Young and R. N. Taylor. Rethinking the taxonomy of fault detection techniques. In *Proc.* 11^{th} *Int'l. Conf. Softw. Eng.*, pages 53–62, Pittsburgh, PA, USA, May 1989.

[20] A. Zeller. Isolating cause-effect chains from computer programs. In *Proc.* 10^{th} *ACM SIGSOFT Symp. Foundations of Softw. Eng.*, pages 1–10, Charleston, SC, USA, Nov. 2002.

[21] A. Zeller. Program analysis: A hierarchy. In *Proc. ICSE'03 Wshop. Dynamic Anal.*, pages 6–9, May 2003.

[22] A. Zeller and R. Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Trans. Softw. Eng.*, 28(2):183–200, Feb. 2002.