Experimental Program Analysis: A New Paradigm for Program Analysis

Joseph R. Ruthruff
Department of Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, Nebraska 68588-0115, U.S.A.
ruthruff@cse.unl.edu

ABSTRACT

Program analysis techniques are used by software engineers to deduce and infer targeted characteristics of software systems for tasks such as testing, debugging, maintenance, and program comprehension. Recently, some program analysis techniques have been designed to leverage characteristics of traditional experimentation in order to analyze software systems. We believe that the use of experimentation for program analysis constitutes a new program analysis paradigm: *experimental program analysis*. This research seeks to accomplish four goals: to precisely define experimental program analysis, to provide a means for classifying experimental program analysis techniques, to identify existing experimental program analysis techniques in the research literature, and to enhance the use of experimental program analysis by improving existing, and by creating new, experimental program analysis techniques.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids, testing tools*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*program analysis*

General Terms

Experimentation

Keywords

experimental program analysis, program analysis, experimentation

1. INTRODUCTION

Program analysis techniques analyze software systems in order to ascertain characteristics of those systems that are of interest to software engineers. The information provided by these techniques can be used to assist researchers and practitioners in testing, debugging, impact analysis, maintenance, program comprehension, and many other software-engineering-related purposes.

In the past, program analysis techniques have generally been considered either "static" or "dynamic". Recent work, however, has gone beyond this traditional classification of program analysis techniques. For example, one of the primary realizations in Delta Debugging [14] is that it does *not* require static information (such as system source code, system specifications, or program dependence graphs) or dynamic information (such as coverage or profile information, or information about runtime exceptions) in advance.

One of the ways that some techniques have moved beyond traditional program analysis considerations is by the use of *experimentation*. For example, one implementation of Delta Debugging, HOWCOME [14], attempts to isolate the minimal relevant variable values in program states in order to explain why a failure f occurred. To do this, HOWCOME applies a subset of variable values in a failing execution to the corresponding variables in a passing execution, and tests whether the applied changes reproduce f. 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. Essentially, HOWCOME is conducting *experiments* designed to isolate the minimal relevant variable values.

Despite this type of recent movement in program analysis, there has been little formal consideration given to the use of traditional experimentation in program analysis. One consequence of this is that many researchers are unaware of experimentation as a potential avenue for addressing program analysis research questions. One effort to account for the use of experimentation in program analysis was undertaken by Zeller [15], who proposed that program analysis techniques reason by means of deduction, observation, induction, and experimentation. In this context, Zeller describes experimentation as a reasoning technique for program analysis, characterized by the use of a tool to control multiple executions of a program in order to determine the causes of observed effects.

We agree that the use of experimentation in program analysis is unique, and warrants separate consideration from other approaches such as static and dynamic analysis. However, its use in program analysis is potentially more extensive than merely the notion of controlling executions with tools, and its goals more powerful than merely explaining cause and effect. Considering the use of experimentation in program analysis, in the context of its traditional use in science, reveals many characteristics of the science of experiment design that either are presently used, or could potentially be used, 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 fa-

ICSE'06, May 20–28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

cilitate the cost-effective study of interactions among multiple factors, and the employment of statistical tests to assess results. The relevance of several of the foregoing characteristics can be seen in debugging, for example. 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 failure. However, there has been little formal consideration of these types of characteristics in program analysis, and the potential for leveraging them, to date, remains largely untapped by the community.

Given the foregoing discussion, this research has four goals. First, this research will precisely define experimental program analysis. Second, this research will provide a means for classifying techniques into an experimental program analysis paradigm. Third, this research will identify a variety of existing experimental program analysis techniques in the research literature. Fourth, this research will enhance the use of experimental program analysis by (1) using approaches in the traditional experiment design literature to improve existing experimental program analysis techniques, and (2) creating new experimental program analysis techniques.

This research offers three primary incentives. First, precise definitions of experimental program analysis will help expose experimental program analysis techniques to the community. Second, the use of traditional experimentation strategies to improve existing experimental program analysis techniques will suggest many opportunities for researchers to improve their own techniques. Third, the creation of new experimental program analysis techniques will promote the use of experimentation in program analysis, thereby empowering researchers to confront program analysis problems in manners that they have not previously considered.

2. 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. [13] introduce an experimental process tailored to the software engineering domain, Fenton and Pfleeger [7] 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. [9] provide guidelines for conducting empirical studies in software engineering. However, these approaches do not formally explain the use of experimentation in program analysis, nor do they provide support for properly classifying experimental program analysis techniques.

There are also instances in which software engineering techniques utilize experimentation 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, 5, 10], and experiment designs are utilized in interaction testing to drive an economic selection of combinations of components to achieve a target coverage level (e.g., [3, 4]). In many cases, we believe that techniques that utilize principles of experimentation may in fact be experimental program analysis techniques. This work will provide support for properly identifying techniques as such.

Within the program analysis domain, to the best of our knowledge, Zeller is the first to have used the term "experimental" in application to program analysis techniques [15]. Zeller's work, however, suffers from three key limitations that this work attempts to address. First, Zeller provides only informal definitions of specific analysis approaches in order to identify their benefits and limitations. In this work, we 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, Zeller's view of experimental program analysis does not consider the rich literature on traditional experimentation and its potential application to the domain of program analysis. In this work, we provide an operational definition for experimental program analysis *in the context* of the foregoing traditional experimentation literature. Third, Zeller's explication contains no discussion of several concepts that are integral to experimentation — e.g., the roles of population and sample selection. He also does not discuss in detail the nature of "control", which requires careful consideration of nuisance variables and various forms of validity threats. All of these quintessentially experimentation-related notions are present in our experimental program analysis definitions, and the utility of including them is supported [11].

3. GOALS AND APPROACHES

In this section, we describe the overall goal of this research, and the approach that will be taken to address its central thesis.

3.1 Overall Goal

The overall goal of the research is to establish a new class of program analysis approaches that are completely experimental in nature, and to provide a staging point for a new area of program analysis research. This research will provide an operational definition for explaining the use of experiments by techniques in order to conduct program analysis, identify techniques that conform to this experimental program analysis definition, and empirically investigate the use of experimental program analysis techniques.

3.2 Activities

The approach for this research involves five activities. After enumerating these activities, we elaborate on how we expect to complete each activity.

1. Define experimental program analysis precisely, and provide a detailed operational definition outlining the traditional experimentation steps performed by experimental program analysis techniques.
2. Identify existing experimental program analysis techniques in the research literature, show how these techniques map to the experimental program analysis operational definition, and use the definition to assess the limitations of the techniques.
3. Use approaches in traditional experiment design to adapt experimental program analysis techniques, and empirically investigate the improvement to the techniques.
4. Design and implement new experimental program analysis techniques to address open problems in the research literature — or to better address program analysis problems with existing, but inadequate, solutions — and empirically evaluate the ability of the new techniques in addressing each targeted problem.
5. If necessary, refine the techniques investigated in Activities 3 and 4, and empirically evaluate the refined techniques.

To accomplish Activity 1, we will descriptively define experimental program analysis in a manner that concisely describes the paradigm. We will augment this descriptive definition with a detailed "operational" definition that will serve as a foundation for outlining the methodologies in traditional experimentation that characterize experimental program analysis techniques. To help

formulate this operational definition, we will draw from an overview of the empirical method that is generalized from various monographs in empirical science. We expect that our operational definition will help map techniques to the experimental program analysis paradigm, thereby serving to help classify techniques.

To accomplish Activity 2, we will survey the broad base of program analysis techniques in the literature, identify techniques that conform to the definitions created in Activity 1, and use these definitions to assess the limitations of the experiments conducted by these techniques. The goal of this survey is two-fold. First, we wish to use our operational definition to show that experimental program analysis is not an abstract theory, but rather a practical paradigm mapping to real program analysis techniques. We have already found well-known and powerful techniques that map to our operational definition, and we expect that an extensive survey of the research literature will reveal additional techniques. Second, we wish to show that our definition is of sufficient power to be extended to a wide variety of program analysis domains. We believe that our operational definition will support the creation of techniques in program analysis domains such as testing, debugging, impact analysis, and program maintenance. This survey will help to identify such domains.

To accomplish Activity 3, we will draw upon research in traditional experimentation to adapt experimental program analysis techniques in an effort to facilitate their improvement. More specifically, from the survey in Activity 2, we will select a variety of techniques from multiple program analysis domains and study whether we can leverage well-established principles from the statistical and experiment design literature to provide improvements related to power, cost-effectiveness, and other metrics. For example, the foregoing literature contains many strategies for sampling large populations of interest in order to support cost-effective investigations, assigning selected treatments to units in the sample to maximize conclusion power while minimizing cost and the influence of extraneous variables, and employing statistical analyses to assess and refine hypotheses under investigation. We expect such opportunities to be explicitly revealed, in part, using the operational definition designed in Activity 1, which maps such experiment design tasks to experimental program analysis techniques.

To accomplish Activity 4, we will identify open problems in the program analysis literature, or problems where existing approaches have not to date provided sufficient solutions, and design new techniques that utilize experimental program analysis to address these problems. We will then empirically evaluate these new techniques to assess their effectiveness, and to compare them with their non-experimental counterparts. We believe that the design, presentation, and evaluation of such techniques will encourage other researchers to consider the use of experimental program analysis.

3.3 Scope and Research Plan

The research just defined will be scoped in a number of ways to make its completion feasible in a reasonable amount of time.

We will limit the number of techniques that will be investigated in Activities 2–4. Adjustments to these activities may be made as we acquire a better understanding of the difficulty of adapting, creating, and empirically evaluating techniques; however, at this time, we expect to identify a minimum of five techniques in Activity 2. We also expect that a minimum of three techniques will be improved in Activity 3, and a minimum of three techniques will be created in Activity 4.

There will also be feedback present within these activities that will influence the process undertaken in other activities. For example, the techniques that are surveyed in Activity 2 may influence those that are adapted in Activity 3, and the domains in which techniques are created in Activity 4; feedback obtained during the empirical evaluations in Activity 3 may influence the evaluations in Activity 4; and so on.

Finally, the five activities will not necessarily progress in a sequential manner. For example, the improvement of existing techniques in Activity 3 may be interleaved with the creation of new techniques in Activity 4. Also, techniques created in Activity 4 may be improved and evaluated in a manner similar to the procedures in Activity 3.

4. PRELIMINARY WORK

We have completed preliminary work toward each of the first four activities listed in Section 3.2. We briefly discuss that work here.

In accordance with Activity 1, we have descriptively defined experimental program analysis as follows:

> *Experimental program analysis* is the evolving process of manipulating a program, or factors related to its execution, under controlled conditions in order to characterize or explain the effect of one or more independent variables on an aspect of the program.

To further elucidate this definition, we have augmented it with a detailed operational definition that serves as a foundation for classifying experimental program analysis techniques, explaining their use of experimentation according to traditional experiment design literature, and assessing their limitations. Due to space constraints, we do not present this operational definition here; instead, we refer readers elsewhere [11].

We have made progress in our survey of the program analysis literature (Activity 2), our improvements of techniques (Activity 3), and our creation of new techniques (Activity 4). In the following discussion, we elaborate on three domains in which we have studied experimental program analysis techniques.

Within the debugging domain, one technique that we have considered is Delta Debugging, which has been used for various debugging-related purposes. For example, the HOWCOME technique [14] uses Delta Debugging to manipulate the variable values in program states, and tests whether those values reproduce the failure of interest, until a minimal subset of variable values is isolated. One improvement that we have proposed to HOWCOME, but not empirically investigated, is to utilize random sampling over the program state space in order to improve the scalability of the technique; using random sampling, 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 chains reported by HOWCOME, 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.

Within the domain of program characterization, one technique that we have investigated is Daikon [6], a technique that infers likely program invariants from execution traces. One improvement that we have made to Daikon is incorporating sequential analysis [12] — a well-known statistical analysis technique — so that it halts the consideration of invariants (and reports them early to the user) after sufficient confidence has been placed in their accuracy. Reporting invariants early may save many unnecessary observations that "further validate" likely invariants. However, it may also increase the number of false-positive invariants reported by the technique. We conducted a case study to investigate the cost-effectiveness of this strategy, where we found that, on average, 16% of the early-reported invariants were false positives. However, in

terms of comparisons between variable values in execution traces to prospective invariants, the savings of reporting invariants early was, on average, 50%. Thus, if performance savings are important, sequential analysis may help improve the practicality of Daikon by ensuring that *some* useful results can be reported, even if the precision of those results suffers.

Within the domain of program optimization, one technique that we have investigated is a new experimental program analysis strategy for automating source code refactoring [8]. Because refactoring a segment of code may not necessarily improve the source code depending on a targeted measure of improvement, and because many types of refactorings are possible, we have proposed an experimental refactoring technique that applies combinations of individual code refactorings to source code, measures their effects, tests whether they should be kept or discarded, and repeats this process until a sufficient number of refactoring groups have failed to improve the program further. We considered an extended example to illustrate the potential of this technique, where we found that experimentally applying code refactorings was preferable to simply applying all possible refactorings, and helped leverage (and address) various interactions between combinations of individual code refactorings that improved (or worsened) the code. To date, we are aware of no well-accepted standards defining when particular code refactorings should (and should not) be applied. Among other things, we believe that an experimental program analysis approach may provide a completely automated means for deciding when and where refactorings should be applied to source code.

5. EXPECTED CONTRIBUTIONS

Experimental program analysis provides numerous opportunities for program analysis and software engineering research. Experimental program analysis 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 experiment designs and sampling techniques that reduce the cost of generalizing targeted aspects of a program. Such advantages will lead to significant advances in program analysis research and in associated software engineering technologies.

We expect that the five activities outlined in Section 3.2 will expose the research community to the use of experimental program analysis techniques, identify domains in which experimental program analysis might be useful, suggest many opportunities for improvements to existing techniques, and promote the use of experimentation in program analysis in order to confront program analysis programs in innovative ways. We also expect many implications from experimental program analysis, even apart from those that we have described here. A first implication involves opportunities for other researchers to create new experimental program analysis techniques. In particular, larger and more difficult program analysis challenges should become reasonable targets with the assistance of, for example, advanced experiment designs and sampling mechanisms. We also anticipate that qualitative improvements to the state of the art will lead to a new generation of experimental program analysis techniques.

A second implication involves automation opportunities with regard to experimentation tasks performed by techniques. It seems likely that the selection of the *approach* for tasks can be automated. For example, experimental program analysis techniques could be encoded to consider multiple experiment designs (e.g., blocking, split-plot, latin square), and select that which is best suited for a specific instance of a problem. Improvements such as these may increase techniques' efficiency, thereby making them more affordable to solve different classes of problems, while freeing human designers from these occasionally difficult and error-prone tasks.

A third implication with somewhat broader potential impacts involves recognizing and exploiting differences between experimental program analysis and traditional experimentation. There are several such interesting differences including, for example, the potential for experimental program analysis 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.

6. 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. Des. Impl.*, pages 168–179, June 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, December 1999.

[3] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Trans. Softw. Eng.*, 23(7):437–444, July 1997.

[4] 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, May 1997.

[5] S. Elbaum and M. Diep. Profiling deployed software: Assessing strategies and testing opportunities. *IEEE Trans. Softw. Eng.*, 31(4):312–327, April 2005.

[6] 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, February 2001.

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

[8] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[9] 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, August 2002.

[10] 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. Des. Impl.*, pages 15–26, June 2005.

[11] J. R. Ruthruff, S. Elbaum, and G. Rothermel. Experimental program analysis: A new program analysis paradigm. Technical Report TR-UNL-CSE-2005-0001, Dept. of Comp. Sci. and Eng., University of Nebraska–Lincoln, April 2005. http://csce.unl.edu/∼ruthruff/papers/tr-unl-cse-2005-0001/TR-UNL-CSE-2005-0001.pdf.

[12] D. Siegmund. *Sequential Analysis: Tests and Confidence Intervals*. Springer-Verlag, 1985.

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

[14] A. Zeller. Isolating cause-effect chains from computer programs. In *Proc.* 10^{th} *ACM SIGSOFT Symp. Found. Softw. Eng.*, pages 1–10, November 2002.

[15] A. Zeller. Program analysis: A hierarchy. In *Proc. ICSE 2003 Wshop. Dyn. Ana.*, pages 6–9, May 2003.