EXPERIMENTAL PROGRAM ANALYSIS

by

Joseph Ronald Ruthruff

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professors Gregg Rothermel and Sebastian Elbaum

Lincoln, Nebraska

May, 2008

UMI Number: 3297759

UMI Microform 3297759

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

EXPERIMENTAL PROGRAM ANALYSIS

Joseph Ronald Ruthruff, Ph.D.

University of Nebraska, 2008

Co-Advisors: Gregg Rothermel and Sebastian Elbaum

Program analysis techniques are used by software engineers to deduce and infer characteristics of programs for software-engineering-related tasks. Recently, some program analysis techniques have been designed to leverage characteristics of traditional experimentation. An examination of these techniques suggests that a new form of program analysis technique can be created by incorporating characteristics of experimentation. To date, however, there has been little formal recognition by the software engineering community of this form of program analysis, or the implications of using it.

This dissertation presents *experimental program analysis* as a unique paradigm for conducting program analysis. We define this paradigm by building on principles and methodologies underlying the use of experimentation in other fields. This work offers four primary contributions. First, we provide definitions of experimental program analysis, illustrate them by example, and describe several intriguing differences between experimental program analysis and the use of experimentation in other research fields. Second, we survey the research literature for experimental program analysis techniques to offer insights into their existence and diversity—both in terms of the techniques themselves and the problem domains in which they operate. Third, we explore the applicability of experimental program analysis in three software engineering problem domains to provide a formative assessment of the capabilities of these techniques, and how they compare—in terms of cost-effectiveness, utility, and

capabilities—to non-experimental baseline techniques. Fourth, we explore the use of experimental program analysis in a large software development setting, and present a case study to investigate the use of experimental program analysis in that setting.

The contributions from the foregoing work support the conjecture that the applicability of experimental program analysis could be substantial. In particular, we expect that this work will expose the research community to 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 new and innovative ways. Accordingly, we believe that the experimental program analysis paradigm offers a promising new direction for program analysis research.

ACKNOWLEDGEMENTS

This work would not have been possible without the assistance and support of many people and organizations.

Gregg Rothermel has served as my undergraduate research advisor, a member of my M.S. committee, and my Ph.D. co-advisor over a period of eight years. He gave me my first research opportunity during my undergraduate studies, introduced me to the topic of experimental program analysis nearly five years ago, and helped me formulate and refine this topic during my doctoral studies. For all of this, I acknowledge and thank him sincerely.

My co-advisor Sebastian Elbaum has provided invaluable intellectual contributions to the development of experimental program analysis during my doctoral studies, and this work would not be where it is today without him. Over the last four years, he has also provided me with personal and professional support for which I am extremely thankful.

I am grateful to Matthew Dwyer and Kent Eskridge for their service on my Ph.D. Committee. I would also like to thank each of them for their capable instruction of courses I have taken at the University of Nebraska–Lincoln, and for their constructive feedback on my dissertation work.

John Penix and David Morgenthaler helped me investigate the experimental screening methodology described in this work. I would like to acknowledge YuQian Zhou, Simon Quellen Field, and Larry Zhou for their work on the static analysis infrastructure at Google, and William Pugh for his assistance in using FindBugs and his feedback on the static analysis factors investigated this work. I would also like to sincerely thank John Penix for supporting my 2007 summer internship at Google Inc., during which the screening methodology work in this dissertation was performed.

I would like to thank Margaret Burnett for advising me during my M.S. studies at Oregon State, for the opportunities she gave me during the two years I worked under her, and for all of her support and advice in the years since.

I thank David Marx for his feedback regarding experimental program analysis in the early stages of the work.

I would like to acknowledge the organizers of the SIR repository, especially Hyunsook Do, Gregg Rothermel, Sebastian Elbaum, and Alex Kinneer. In particular, Hyunsook Do advised me during the preparation of some of the software artifacts used in this dissertation's empirical studies. I also thank all the researchers who have contributed to the SIR artifacts utilized in this work.

My fellow students in the ESQuaReD Laboratory at the University of Nebraska–Lincoln have made the last four years an enjoyable experience. The ESQuaReD faculty have created and fostered a constructive and engaging research environment that it has been my honor to be a part of. To all of the past and present ESQuaReD students and faculty that I have interacted with, thank you.

My closest friends have provided me with immeasurable support and, shall we say, good humor throughout the last 10 years. These years would not have been the same without them. They know who they are, and I offer each of them many thanks.

My family has been a constant source of love and support for me, and I would not be where I am today without them. In particular, this culmination of my education would not have been possible without my father Ron, my mother Kathy, and my brother Bill. To all of you, thank you so much.

Finally, I would like to thank Kristen, for everything. In particular, she has supported and sustained me during the latter half of my doctoral studies in so many ways, particularly during the final months of my studies, and has added the type of happiness, joy, and love into my life that I never thought I would find.

This work has been supported in part by the National Science Foundation under awards CNS-0454203, CCF-0440452, and CCR-0347518 to the University of Nebraska–Lincoln, by the University of Nebraska–Lincoln through an Othmer Fellowship and the Jensen Chair of Software Engineering, and by the Test Engineering and Static Analysis groups at Google Inc. The opinions and conclusions within this dissertation are my own, and do not reflect the views of any of the foregoing organizations.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contributions of and Incentives for Research . . . 4
1.3 Outline . . . 5

2 Background **8**
2.1 Experimentation and Empirical Research . . . 8
2.1.1 A Generalization of Traditional Experimentation . . . 9
2.1.2 Validity Threats in Experiments . . . 12
2.2 Program Analysis . . . 17

3 Experimental Program Analysis **20**
3.1 Descriptive Definition . . . 20
3.1.1 Definition of an Experiment . . . 21
3.1.2 Definition of Experimental Program Analysis . . . 23
3.2 Operational Definition . . . 27
3.2.1 Motivation and Goals . . . 27
3.2.2 Tasks to Implement Features in Experimental Program Analysis Techniques . . . 31
3.2.3 Summary of Support Offered by Operational Definition . . . 51
3.2.4 Relating the Descriptive Definition to the Operational Definition 56
3.3 Distinguishing Traits of Experimental Program Analysis . . . 60
3.3.1 Replicability and Sources of Variation . . . 60
3.3.2 Sampling the Input Space . . . 62
3.3.3 The Cost of Applying Treatments . . . 63
3.3.4 Assumptions about Populations . . . 65
3.3.5 Procedures versus Algorithms . . . 66
3.3.6 The Role of Feedback . . . 67
3.4 Related Work . . . 68
3.4.1 Experimentation in Program Analysis . . . 68
3.4.2 Experimentation in Software Engineering Research . . . 70
3.4.3 Search-based Software Engineering . . . 71
3.4.4 Static and Dynamic Analysis . . . 72

4 Survey of Experimental Program Analysis Techniques 75
4.1 Techniques Conforming to Experimental Program Analysis 78
4.1.1 Delta Debugging: Minimizing Failure-Inducing Input 78
4.1.2 The Chaining Technique for Test Data Generation 82
4.1.3 Continuous Compilation 87
4.1.4 Signature Extraction from Malware Detectors 92
4.1.5 Dynamic Detection of Program Invariants 96
4.2 Techniques Not Conforming to Experimental Program Analysis . . . 101
4.2.1 Profiling Software 102
4.2.2 Program Testing 103
4.2.3 Finding Failure-inducing Changes using Change Classification 106
4.2.4 Inferring Faults in Program Code 107
4.3 Discussion 109
4.3.1 Analysis Amenable to Experimental Program Analysis 109
4.3.2 Summary of Survey Findings 111

5 Investigation of Three Applications of Experimental Program Analysis 113
5.1 Experimental Program Refactoring 114
5.1.1 Problem Domain: Program Transformation 114
5.1.2 Experimental Program Analysis Technique 117
5.1.3 Related Work 122
5.1.4 Case Study 123
5.2 Experimental Regression Fault Analysis 132
5.2.1 Problem Domain: Program Debugging 132
5.2.2 Experimental Program Analysis Technique 134
5.2.3 Related Work 140
5.2.4 Case Study 141
5.3 Experimental Dynamic Invariant Detection 148
5.3.1 Problem Domain: Program Understanding 148
5.3.2 Improving Cost-effectiveness in an Experimental Program Analysis Technique 150
5.3.3 Case Study of Cost-effectiveness Improvement 156
5.3.4 Evaluating Statistical Assumptions Made by an Experimental Program Analysis Techniques 163
5.3.5 Case Study of Testing Distribution Assumptions Improvement 169

6 Experimental Program Analysis in the Real World 177
6.1 Setting: Static Analysis Tools 178
6.1.1 Inherent Challenges in using Static Analysis Tools 178
6.1.2 Deployment of the FindBugs Tool at Google 179
6.1.3 A New Approach: Automatically Classifying Incoming Static Analysis Warnings 181

6.2 Building Binomial Logistic Regression Analysis Models 182
6.2.1 Background: Regression Analysis 182
6.2.2 Design Variables for Logistic Regression Models 185
6.3 Experimental Screening Methodology 191
6.3.1 Screening Experiments in a Program Analysis Context 194
6.3.2 Screening Methodology 197
6.3.3 Threats to Validity in the Experimental Program Analysis Technique 200
6.4 Example of Building Models From Screening Factors 201
6.4.1 Model for Predicting False Positives 202
6.4.2 Models for Actionable Warnings 205
6.4.3 Discussion: Construction of Models 208
6.5 Related Work 209
6.6 Case Study 211
6.6.1 Design 211
6.6.2 Results and Discussion 219
6.7 Discussion: Generalizing Experiences 227
6.7.1 Use of Classification Models 228
6.7.2 Multi-dimensionality of Data 229
6.7.3 Abstraction of Programs and Related Entities 230

7 Conclusions and Future Work **232**
7.1 Merit and Impact 232
7.2 Future Work 234

Bibliography **237**

List of Figures

3.1 Mapping the features of the operational definition to the key components of the descriptive definition. 57

5.1 An algorithmic representation of the experimental program refactoring methodology. 120
5.2 An algorithmic representation of the experimental regression fault analysis methodology. 137
5.3 An illustration of using a fractional design to distribute treatments execution trace data across sampled program points for Daikon. . . . 154

6.1 An overview of the use of screening at Google to identify factors for building classification models for static analysis warnings. 194
6.2 An algorithmic representation of the experimental screening methodology. 198
6.3 The elimination of factors for predicting false positive warnings through screening. 204

List of Tables

3.1 An operational definition of experimental program analysis. 29
3.2 Mapping HOWCOME to the operational definition of experimental program analysis. 53

4.1 Summary of survey of experimental program analysis techniques in research literature. 77
4.2 Mapping the Delta Debugging (DD) technique simplifying failure-inducing input to the experimental program analysis operational definition. . . 79
4.3 Mapping the Chaining technique for test generation to the experimental program analysis operational definition. 84
4.4 Mapping the Continuous Compilation methodology to the experimental program analysis operational definition. 88
4.5 Mapping the malware detector signature extraction technique to the experimental program analysis operational definition. 94
4.6 Mapping Daikon to the experimental program analysis operational definition. 97

5.1 Mapping the experimental refactoring approach to the experimental program analysis operational definition. 118
5.2 The Lack of Cohesion of Methods of classes using baseline approaches and experimental refactoring. The * symbol indicates that no refactorings were identified for this class. 127
5.3 The abstractness of the `Siena` package using baseline approaches (INIT and ALL) and experimental refactoring (EPA). 129
5.4 Summary of the utility of all code refactorings, and interactions between beneficial or neutral refactorings causing detrimental effects. . . 130
5.5 The number of refactorings that need to be applied and tested using an exhaustive search versus experimental program analysis (EPA). . . 130
5.6 Mapping the experimental regression fault analysis approach to the experimental program analysis operational definition. 135

5.7 Summary of the number of isolated changes, in terms of functions and lines of code, responsible for the seeded `flex` fault using five test cases. The cost of identifying these changes is also shown in terms of the number of evaluations performed by the technique and time (Minutes:Seconds for the experimental regression fault analysis approach and Days:Hours:Minutes:Seconds for the exhaustive approach). 145
5.8 Mapping Daikon with a fractional design to the experimental program analysis operational definition. 155
5.9 The number of test cases in five randomly-selected `Space` test suites. 158
5.10 Results of using a fractional design in Daikon. 160
5.11 Per-test-suite results regarding the number of additional false positive invariants reported, and observations saved during analysis, for `Space`. 160
5.12 Mapping Daikon with goodness-of-fit tests for verifying distributional assumptions to the experimental program analysis operational definition. 167
5.13 Justified invariants rejected by Kolmogorov-Smirnov goodness-of-fit tests using 100% of the execution trace data. 173
5.14 The number of invariants retained by Daikon$_{KS}$ when provided with 75% and 100% of the available execution traces. 173
5.15 Number of retained and additionally rejected false positive invariants for the original implementation of Daikon when provided with 50%, 75%, and 100% of the available execution traces. 175

6.1 Factors for design variables of logistic regression models. 188
6.2 Mapping the experimental screening methodology to the experimental program analysis operational definition. 196
6.3 Factors selected through screening for false positive prediction models. Coefficients for pattern are not shown because there are 156 coefficients for this factor. 204
6.4 Factors selected for actionable prediction models considering only true defects. Coefficients for pattern are not shown. 206
6.5 Factors selected for actionable prediction models considering all warnings. Coefficients for pattern are not shown. 207
6.6 Relating factors from Ostrand et al. [110] and Bell et al. [20] to similar factors in this work. 213
6.7 Cost of gathering data for the four types of models in terms of time (Days:Hours:Minutes:Seconds). Data is only gathered once for the `All-Data`, `BOW`, and `BOW+` models. 220
6.8 Cost of building the models in R, in terms of time (Minutes:Seconds) using each of the given "Model Type" data sets. 220
6.9 The accuracy of the classification models in predicting false positive warnings. For the holdout data evaluation, the average precision of the models from the three observations is shown. 222

6.10 The accuracy of the classification models in predicting actionable warnings. Actionable warnings are predicted using both only true defects, and all warnings—including false positives. For the holdout data, the average precision of the models from the three observations is shown. 223
6.11 The three precision observations for the models predicting false positives using holdout data. 223
6.12 Predicting actionable warnings. 224
6.13 The R^2 values for the `Screening` and `All-Data` models. 224

Chapter 1

Introduction

1.1 Motivation

Program analysis techniques analyze programs[1] to collect, deduce, or infer specific information about those programs. The resulting information typically involves program properties and attributes 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. Researchers who are investigating these and other activities continue to seek new program analysis techniques that can address software engineering problems cost-effectively, and to seek ways to improve existing techniques.

Recently, researchers have begun to consider a new approach to program analysis that harnesses principles of experimentation [129, 155]. Zeller [155], for example, describes a hierarchy of four reasoning techniques that, he argues, can be utilized by program analysis techniques—one of these reasoning techniques being *experimen-*

[1] In this dissertation, we use the term "program" to refer to a software system.

tation.[2] The use of principles of experimentation and empirical research methods in program analysis contexts in this prior work might support the use of program analysis techniques that are capable of drawing inferences about properties and attributes of programs, characterizing aspects of programs, and establishing causality relationships between program variables of interest in cases in which more traditional analyses have not succeeded. For example, Zeller describes his HOWCOME technique [154], which investigates the states of a program execution to isolate the variable values responsible for failures,[3] as a "purely experimental" technique [154, page 1] and writes that it is "a novel and very different approach" from "(traditional) approaches to facilitate debugging (that) have relied on static or dynamic program analysis" [154, page 1].

As experimentalists, we do indeed recognize many characteristics of 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 use of inferential analyses to establish degrees of confidence in 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

[2] On Zeller's view, such experimentation is characterized by a search for cause-effect relationships, but we shall see in this dissertation that a wider view can be profitably taken.

[3] Following standard terminology [15], in this dissertation, we refer to an incorrect computational result as a *failure*, and the incorrect part of the program that caused the failure as a *fault*.

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 such as Delta Debugging [29, 31, 154, 156], statistical debugging [90, 91], and interactive testing-based fault localization techniques [128, 130, 131, 134].

While the use of experimentation in program analysis techniques has increased in recent years, there are many domains in program analysis where experimentation could be further used to make advances in program analysis techniques, but to date has not been used to its full potential. As we shall see later in this dissertation, there are also many opportunities for existing program analysis techniques to draw upon the decades of research in the statistical and experiment design literature to improve their efficiency or effectiveness by offering (1) methodologies for manipulating independent variables of interest to test their effects on dependent variables, (2) procedures for conducting and adjusting hypothesis tests in program analysis contexts, (3) strategies for systematically controlling sources of variation during these tests, (4) experiment designs and sampling techniques to reduce the costs of experimentation, and (5) mechanisms to generate confidence measures in the reliability and validity of the results. To date, however, these opportunities have not been pursued rigorously by the research community. As a result, we conjecture that many program analysis techniques and domains have not advanced to the degree that they could have had their experimental nature been better understood and exploited by researchers.

1.2 Contributions of and Incentives for Research

In this work, we argue that a class of program analysis approaches exists whose members are inherently experimental in nature. By this, we mean that these techniques can be characterized in terms of guidelines and methodologies defined and practiced within the long-established paradigm of scientific experimentation. Building on this characterization, we define *experimental program analysis* as a unique program analysis paradigm. We show how program analysis techniques can be characterized in terms of this paradigm, and how the study of these techniques in light of the existence of this paradigm can lead to several promising avenues of research.

This dissertation presents the work that was undertaken in an attempt to demonstrate the existence and utility of experimental program analysis as a paradigm for program analysis. The contributions of this work, enumerated in the order of their presentation in this dissertation, are as follows:

1. We present definitions of experimental program analysis, and discuss in detail how principles from the statistical, experimental design, and empirical research methods literature relate to the experimentation-based tasks performed by experimental program analysis techniques.

2. We present a survey of the research literature that identifies several existing experimental program analysis techniques. We show how these techniques map to our definitions of experimental program analysis, and use the definitions to assess the limitations of the surveyed techniques. We also show how the support offered by these definitions makes the process of identifying these limitations easier.

3. We study three experimental program analysis techniques that illustrate the range of experimental program analysis, both in terms of its applicable do-

mains and the capabilities of the techniques themselves. We also study these techniques empirically, evaluating their cost-effectiveness, utility, and capabilities compared to non-experimental baseline techniques.

4. We explore the use of experimental program analysis in a large software development setting, and show how experimental program analysis can be used to address a real-world problem in this setting. We empirically explore the underlying principles governing the use of experimental program analysis and account for its success (or lack thereof) in this domain.

This research offers three primary incentives and areas for potential impact. First, a definition of experimental program analysis will serve to inform the research community about the nature of these types of program analysis techniques. It will enable researchers to better understand existing techniques that fall within this domain, how these techniques operate, and why they can be successful. Second, a definition of experimental program analysis and an elucidation of the statistical and empirical research methods literature and principles surrounding this paradigm will suggest opportunities for improving existing techniques by enhancing the sophistication of the experiments they conduct. Third, a demonstration of the utility of experimental program analysis in important program analysis and software engineering settings, we believe, will suggest and spur the creation of new experimental program analysis techniques, thereby empowering researchers to confront research problems in manners that they have not previously considered.

1.3 Outline

The remainder of this dissertation proceeds as follows. Chapter 2 overviews fundamental principles in experiments and empirical research methods that are pertinent

to the understanding of experimental program analysis. This chapter also discusses some directions in program analysis research that are related to our formulation of experimental program analysis.

Chapter 3 presents experimental program analysis as a paradigm for program analysis. This chapter introduces our definitions of experimental program analysis, and elaborates in detail how the key components of experiments are conducted by experimental program analysis techniques in a program analysis context. This chapter also discusses important aspects of experimental program analysis techniques in comparison to experimentation in other problem domains, and reviews work in the research literature that is related to experimental program analysis.

Chapter 4 presents a survey of experimental program analysis in the research literature. This survey discusses several techniques that conform to our definition of experimental program analysis, and shows in detail how these techniques can be classified as experimental program analysis techniques. This survey also discusses techniques that display some characteristics of experimentation, but cannot be classified as experimental program analysis.

Chapter 5 presents and studies three experimental program analysis techniques in detail. Two of these techniques are introduced in this dissertation, while the third technique has been previously studied in the research literature but, here, is modified in an attempt to improve the experiments conducted by the technique. This chapter empirically studies each of these three program analysis techniques to explore the utility (if any) of using experimental program analysis as compared to non-experimental techniques.

Chapter 6 presents an in-depth case study regarding the use of experimental program analysis in a large software development environment. This chapter presents a new experimental program analysis technique that was incorporated into the work-

flow of a large software company, Google Inc., and empirically studies the costs and benefits of using the technique over a period of months.

Finally, Chapter 7 concludes this dissertation by summarizing the merit and impact of this work, and discussing opportunities for future work.

Chapter 2

Background

This chapter overviews the material necessary to understand experimental program analysis. Because experimental program analysis is rooted in both experimentation and program analysis, we overview material in both these areas.[1]

2.1 Experimentation and Empirical Research

The field of empirical research methods is mature, and has been well-discussed in various research monographs (e.g., [22, 83, 102, 119, 138, 144, 147, 149]). Because experimental program analysis techniques draw on empirical research methods by conducting experiments in order to perform program analyses, in this section we distill, from these monographs, an overview of the empirical method. In addition, we highlight material about experiments that will be relevant to the understanding of experimental program analysis. (Note that this overview is generalized from the sources just cited. In practice, the empirical method takes on different forms in

[1]Portions of the material presented in this chapter, Chapter 3, and Chapter 5 have appeared previously in [132].

different domains. The generalization we present here, which we hereafter refer to as "traditional experimentation," suffices for our subsequent discussion.)

2.1.1 A Generalization of Traditional Experimentation

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, or limitations on those outcomes. In the scientific arena, research questions can be exploratory, descriptive, or explanatory in nature—attempting not just to establish causality but, more broadly, to establish relationships and characterize a population [83, 102, 119, 144]. (In Chapter 3, we discuss how these outcomes are possible with experimental program analysis techniques as well.) 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, different research strategies (e.g., case studies, surveys, experiments) may be employed to answer the formulated research questions. Conditions for selecting strategies may include the desired level of control over extraneous factors, the available resources, and the need for generalization. These strategies have different features and involve different activities. In the case of experiments—the design strategy of interest in this dissertation—scientists seek to *test hypothesized relationships between independent 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