


2015

The Effect of Applying Design of Experiments Techniques to Software Performance Testing

Gloria Johnson
Walden University

Follow this and additional works at: <https://scholarworks.waldenu.edu/dissertations>

 Part of the [Business Administration, Management, and Operations Commons](#), and the [Management Sciences and Quantitative Methods Commons](#)

This Dissertation is brought to you for free and open access by the Walden Dissertations and Doctoral Studies Collection at ScholarWorks. It has been accepted for inclusion in Walden Dissertations and Doctoral Studies by an authorized administrator of ScholarWorks. For more information, please contact ScholarWorks@waldenu.edu.

Walden University

College of Management and Technology

This is to certify that the doctoral dissertation by

Gloria Johnson

has been found to be complete and satisfactory in all respects,
and that any and all revisions required by
the review committee have been made.

Review Committee

Dr. Branford McAllister, Committee Chairperson,
Applied Management and Decision Sciences Faculty

Dr. Robert Kilmer, Committee Member,
Applied Management and Decision Sciences Faculty

Dr. Christos Makrigeorgis, University Reviewer
Applied Management and Decision Sciences Faculty

Chief Academic Officer
Eric Riedel, Ph.D.

Walden University
2015

Abstract

The Effect of Applying Design of Experiments Techniques to Software Performance

Testing

by

Gloria Johnson

MBA, University of Dallas, 1995

BS, Jackson State University, 1975

Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Applied Management and Decision Sciences

Walden University

January 2015

Abstract

Effective software performance testing is essential to the development and delivery of quality software products. Many software testing investigations have reported software performance testing improvements, but few have quantitatively validated measurable software testing performance improvements across an aggregate of studies. This study addressed that gap by conducting a meta-analysis to assess the relationship between applying Design of Experiments (DOE) techniques in the software testing process and the reported software performance testing improvements. Software performance testing theories and DOE techniques composed the theoretical framework for this study. Software testing studies ($n = 96$) were analyzed, where half had DOE techniques applied and the other half did not. Five research hypotheses were tested, where findings were measured in (a) the number of detected defects, (b) the rate of defect detection, (c) the phase in which the defect was detected, (d) the total number of hours it took to complete the testing, and (e) an overall hypothesis which included all measurements for all findings. The data were analyzed by first computing standard difference in means effect sizes, then through the Z test, the Q test, and the t test in statistical comparisons. Results of the meta-analysis showed that applying DOE techniques in the software testing process improved software performance testing ($p < 05$). These results have social implications for the software testing industry and software testing professionals, providing another empirically-validated testing methodology. Software organizations can use this methodology to differentiate their software testing process, to create more quality products, and to benefit the consumer and society in general.

The Effect of Applying Design of Experiments Techniques to Software Performance
Testing

by

Gloria Johnson

MBA, University of Dallas, 1995

BS, Jackson State University, 1975

Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy
Applied Management and Decision Sciences

Walden University

January 2015

Dedication

I dedicate this study to my faithful champion, my son, Kevin. I am deeply grateful for the tenacity and consistency of your encouragement and support.

Acknowledgments

First and foremost, all the glory, honor, and praise go to God for getting me through this program. I would like to acknowledge and thank my committee for all their guidance and patience throughout this process. Thank you to Dr. Branford McAllister, Dr. Robert Kilmer, and Dr. Christos Makrigeorgis. Thank you Dr. Teresa Bittner for your encouragement as I started this journey. I am deeply grateful to Cheryl Lockett for the help, encouragement, and unwavering friendship throughout this doctoral journey. To my Walden colleagues, thank you for your encouragement, advice, and support during our time together on this doctoral journey. To my son, my number one supporter, my extended family, friends, and church family, my deepest gratitude for your prayers and unfailing support. Finally, thank you Walden faculty and staff for your expertise, advice, and guidance.

Table of Contents

List of Tables	vi
List of Figures	viii
Chapter 1: Introduction to the Study.....	1
Background	1
Purpose of the Study	8
Nature of the Study	9
Research Questions and Hypotheses	10
Research Questions.....	11
Research Hypotheses	12
Theoretical Bases for the Study	14
Definitions of Terms and Acronyms.....	15
Assumptions, Limitations, Scope, and Delimitations.....	19
Assumptions.....	19
Limitations	19
Scope	19
Delimitations.....	20
Significance of the Study	20
Gap in Current Literature.....	21
Professional Application.....	21
Implications for Positive Social Change.....	22
Summary.....	22

Chapter 2: Literature Review	24
Introduction.....	24
Literature Search.....	25
Search Strategy	25
General Theoretical Concepts.....	27
Software Testing	27
DOE Strategies.....	32
Meta-Analysis	38
Literature Examination and Analysis.....	49
Early Prior Research	49
Current Research.....	50
Variables of Interest	55
Summary	56
Chapter 3: Methodology and Design	59
Introduction.....	59
Target Population.....	60
Sampling Design.....	61
Sampling Eligibility Criteria.....	62
Sample Size Calculation	64
Research Design and Method	66
Variables of Interest Format Definitions	67
Data Collection Procedure	69

Coding.....	70
Effect Size and Data Computations	73
Variables and Hypotheses.....	74
Variables	75
The Hypotheses.....	78
Testing the Hypotheses	82
Data Analysis	83
Fixed Effects Versus Random Effects	84
Statistical Analysis.....	85
Presentation of Results.....	85
Publication Bias	88
Summary	90
Chapter 4: Research Results	91
Introduction.....	91
Data Collection and Preparation	97
Data Characterization.....	98
Publication Bias	100
Data Coding Scheme.....	104
DOE Techniques.....	108
Effectiveness Measures.....	108
Testing Duration	109
Software Testing Settings	110

Testers Proficiency.....	110
Publication Timeline.....	111
Testing Phase	111
Publication Type	111
Data Analysis	112
Assessing The Hypotheses.....	115
Defects Detected	117
Defects Detection Rate	120
Phase Detected	122
Testing hours.....	125
Key Findings.....	127
Summary	133
Chapter 5: Discussion, Conclusions, and Recommendations.....	136
Overview.....	136
Interpretation of Findings	138
Limitations of the Research Study	142
Threats to Validity	142
Threats to Generalizability.....	142
One Number Summarization of a Research Study	143
File Drawer Problem.....	144
Mixing Apples and Oranges	144
Recommendations for Future Research	144

Software Testing Publication Bias.....	145
Software Testing Effectiveness Measurement.....	146
Software Testers.....	146
Implications for Social Change.....	147
Potential Societal Impact	148
Potential Impact for the Software Testing Industry	148
Potential Impact for the Software Testing Professionals	149
Conclusion	149
References.....	152
Appendix A: Statement to the Validation of CMA Version 2.....	172
Appendix B: Comprehensive Meta-Analysis Version 2 Testimonials	173
Appendix C: Included Original Studies Without DOE Techniques	175
Appendix D: Included Studies With DOE Techniques	180
Appendix E: Cumulative Statistics for Studies where DOE was applied.....	186
Appendix F: Cumulative Statistics for Studies where DOE was not applied.....	190
Appendix G: Formulas Used in the CMA Package Computations.....	193
Appendix H: Raw Data.....	195
Appendix I: Effectiveness Measures Statistical Data	202
Curriculum Vitae	210

List of Tables

Table 1. Hypothesis Makeup by Number of Original Studies..... 12

Table 2. A Framework for Choosing the Appropriate DOE Strategy 37

Table 3. Sample of Software Testing Studies Characteristics That Can Be Coded 71

Table 4. Software Testing Application Coding Sample 72

Table 5. Effect Size Magnitude Rule of Thumb 87

Table 6. Variables of Interest..... 105

Table 7. Coding Scheme 107

Table 8. Fixed effect model: Overall Results 116

Table 9. Moderator Analysis: Defects Detected Summary Statistics 118

Table 10. Statistics for t test on Defects Detected Data..... 119

Table 11. Moderator Analysis: Defect Detection Rate Summary Statistics 121

Table 12. Statistics for t test on Defect Detection Rate Data..... 121

Table 13. Moderator Analysis: Defects Detected By Phase Summary Statistics 123

Table 14. Statistics for t test on Defects Detected By Phase Data..... 124

Table 15. Moderator Analysis: Testing Hours Summary Statistics..... 126

Table 16. Statistics for t test on Testing Hours Data 127

Table 17. Summary of Study Findings for Effectiveness Measures..... 131

Table 18. Research Results Summary 134

Table C1. References Without Design of Experiments 175

Table D1. References With Design of Experiments 180

Table E1. Data Characteristics Raw Data for Studies That Applied DOE Techniques . 186

Table F1. Data Characteristics Raw Data for Studies That Did Not Apply DOE	
Techniques	190
Table H1. Studies Without Design of Experiments Raw Data Calculations	195
Table H2. Studies With Design of Experiments Raw Data Calculations	199

List of Figures

Figure 1. Meta-analysis results showing funnel plots and computed statistics for a fixed-effect model.	89
Figure 2. Year of publication for the included studies.....	99
Figure 3. Publication profiles of original studies according to study setting.....	100
Figure 4. Funnel plot indicating the possibility of publication bias.	102
Figure 5. Precision funnel plot with collected and imputed data.....	104
Figure E1. Forest plot of original included studies that had DOE techniques applied. ..	189
Figure F1. Forest plot of original included studies that did not have DOE techniques applied.....	192
Figure I1. Computed statistical data for studies reporting findings as detected defects.	202
Figure I2. Forest plot studies reporting findings as detected defects.....	203
Figure I3. Computed statistical data for studies that reported defect detection rate.	204
Figure I4. Forest plot for studies reporting findings as defect detection rate.	205
Figure I5. Computed statistical data for studies that reported defects by phase detected.	206
Figure I6. Forest plot for studies that reported defects by phase detected.....	207
Figure I7. Computed statistical data for studies that reported total testing hours.....	208
Figure I8. Forest plot for studies that reported total testing hours.....	209

Chapter 1: Introduction to the Study

This chapter begins with a discussion of the background for software and software testing to form the framework for the problem with software testing effectiveness. The problem statement and the purpose are followed by discussions of the research approach and this study's implications for positive social change.

Background

The fast pace of technological changes driving the Internet, social networking, improved user interfaces, faster computer hardware, and more affordable computer hardware has resulted in more and more people in today's society becoming computer literate. Desktop computers, laptop computers, and computer-based products are as common in homes as are televisions. Additionally, computers and computer software are at the center of almost anything, from household appliances to home security networks to cell phones to children's toys to automobiles. Consequently, the demand for software and computer software products has grown and continues to grow.

Software development organizations are no longer found only in businesses whose core competencies are based on computers or computer software. Software development organizations, in virtually every business domain in today's society (for example, manufacturing, medical, defense industry, and services) are faced with satisfying this ever-increasing demand for more innovative software and software products. Moreover, the increase in the demand on software can be linked to hardware improvements, changes in computing architecture, and increases in memory and storage capacity, as stated by Gupta, Kapur, and Jha (2008). With this demand comes the

responsibility for these organizations to deliver highly reliable, quality software and software products.

In an effort to meet this increasing demand for reliable software and software products of the highest quality, software organizations have begun to concentrate as much on software testing as on the design and development of the software. A couple reasons for this emphasis on software testing are life-affecting products controlled by software and efforts to prevent software defects from being discovered by customers. Lazić (2010) suggested that software defects discovered by the customer after product delivery incur the heaviest defect-removal costs. Hence, software quality managers constantly seek solutions to improve testing effectiveness, reduce testing costs, and reduce test time, according to Nirpal and Kale (2012). The entire software development life cycle is important to the generation of quality software and software products, but the fact that the software testing phase consumes a large portion of the software development budget makes it a particularly critical phase of the software development life cycle.

Lazic and Velasevic (2004) estimated that 30% to 70% of a software development budget is typically spent on testing. This gives software testing a particularly crucial role in defining the final software product's quality; hence, it is not unusual to dedicate at least 50% of project resources to this phase (Sagarna & Lozano, 2005). Similarly, Nirpal and Kale (2012) also suggested that testing costs often account for up to 50% of the total software development costs. Kadry and Kalakech (2011) further noted that software development companies spend more time on maintenance of existing software than on development of new software, basing their opinion on earlier studies showing software

maintenance accounts between 40–70% of the total life-cycle costs. Researchers posited that these assertions regarding software testing costs still hold and tended to agree that the proportional costs of the software testing phase have remained constant since the surveys conducted in the 1970s and 1980s (Lazić, 2010). If this has become the status quo for software testing, it is not surprising that software development organizations continue to place more and more emphasis on the testing phase.

Traditionally, requirements-based test case design together with intuition-based test case selection approaches (i.e., based on the whims, background, and experience of the tester) have been the norm for software testing (Qin & Wang, 2009). Interestingly enough, the research efforts in the software testing community have focused on new technology and new software testing tools for improving software performance testing. Two examples of such research efforts are projects conducted by the federal government agency, The National Institute of Standards and Technology (NIST): Automated Combinatorial Test for Software (ACTS) and Covering Arrays Research papers resulting from such projects by NIST scientists include Kuhn, Kacker, and Lei (2008) and Kuhn, Kacker, and Lei (2009).

Considering the software testing results from these projects as evidenced by the published articles, the traditional requirements-focused testing approaches might not have been the best testing techniques for producing high quality software products. Unfortunately, testing professionals have encountered many problems using these traditional approaches. Some of the challenges facing test professionals using traditional

requirements-focused approaches, as cited by Rao and Sastri (2011), include the following:

- Testers incorrectly interpreting the requirements.
- Making unfounded assumptions while testing.
- Testers lacking sufficient domain knowledge.
- Testers being too dependent on the developers to understand the requirements.
- Testers not directly involved with gathering customer requirements and often becoming involved later in the development cycle.

These problems adversely impact software testing not only in terms of cost but also in terms of the accuracy of the test results, which ultimately affect cost. From this list, it can be seen that there are benefits to be gained from moving away from the commonly-practiced, intuition-based testing. Many of the problems Rao and Sastri (2011) observed could be addressed by adopting combinatorial testing approaches that are (a) less tester-focused, (b) powerful, (c) repeatable, and (d) focused on using appropriate test tool(s) selected from the considerable availability of user-friendly tools on the market.

Seemingly, efforts to improve software performance testing may prove more fruitful if the test case design were based on a better understanding of the interactions of selected software test factors chosen to achieve the greatest testing coverage.

Experimental design techniques support a methodology to achieve optimal testing coverage. This premise of understanding test factor interactions possibly leading to

software testing performance improvements formed the foundation for the question in this research.

Design of experiments (DOE) refers to a systematic approach to planning an investigation so that the appropriate data are collected and statistically analyzed, according to Montgomery (2009). Montgomery purported DOE to be one of the most powerful tools available for the design, characterization, and improvement of products and services. Experimental designs have been used to plan, conduct, and analyze testing to obtain the optimal performance for a system or process using the minimum input data or process steps. DOE techniques offer a systematic approach to the investigation of a system or process by utilizing tests designed in such a manner that planned changes are made to the input variables to a process or system. The effects of these changes on a predefined output are then assessed.

Additionally, Antony, Chou, and Ghosh (2003) posited that mathematically-based experimental designs allow for an objective conclusion based on the statistical significance of input variables, either acting alone or in combination with one another. The statistical, iterative approach of experimental designs gives the methodology an advantage over the *one change at a time* experimental methods, in which input variables interactions cannot be observed. With one change at a time to input variables, researchers run the risk of observing a seemingly significant result only to have the result nullified when variable interactions are observed.

To apply the statistical approach to designing and analyzing experiments, Montgomery (2009, p. 14) recommended guidelines for the procedure as follows:

1. Recognize and clearly state the problem.
2. Select the response variable.
3. Choose the factors, factor levels, and ranges.
4. Choose the experimental design.
5. Perform the experiment.
6. Statistically analyze the data.
7. Derive conclusions and make recommendations.

In experimental designs, changes are made to input variables so that the reasons for the changes in the output responses are observed and identified. The step where the experiment is actually performed is an iterative step. The preplanning steps are considered Steps 1 through 3. According to Montgomery, the success of this methodology hinges on how well the first three steps are performed. Starting experimental designs with proper planning and setup are important for ending up with repeatable, valid, and verifiable results.

Theories have been offered that link effective software performance testing to the application of DOE techniques used to plan and design test cases. Prior investigative research efforts related to this area of inquiry, which are discussed in detail in the literature review presented in Chapter 2, have assessed the impact the application of DOE techniques has on software performance testing. Findings have shown evidence of testing performance improvements. For example, Raske (1994) showed that applying DOE methodology to a set of relevant software factors could result in the design of the test suite composed of the minimum amount of tests needed to assess software testing

performance effectiveness. Bandurek (2005) studied DOE techniques in product validations while Gupta and Jalote (2008) proposed a mathematical approach for experimentally evaluating software performance testing.

Additionally, utilizing DOE techniques, a researcher can plan for all possible dependencies and then stipulate exactly what data are needed to assess whether the input variables alter the resulting response on their own, when combined in interactions with other variables, or whether there is a change at all (Montgomery, 2009). As shown in Table D1, I used experimental designs in this research to test the significance of the correlation between the DOE techniques applied in test case selection and the resulting effectiveness of the software performance testing. I used the research design methodology, meta-analysis, to investigate the same assertions as the group of original studies included in this research project.

Statement of the Problem

In recent years, there has been an increase in research efforts investigating the hypothesis that the application of DOE techniques in test case design improves the efficiency and effectiveness of software performance testing. Individual, isolated single research efforts have reported findings to support such testing performance improvements. The current scholarly research literature of reported findings seemed to report only single research efforts. There is a gap in the current literature of reported findings for a group of such studies. There is a gap in the scholarly research literature of concerted, concentrated efforts quantitatively to validate measurable software testing performance improvements with objective statistical data across a group of selected

studies. The findings from isolated individual studies provide insufficient scientific evidence of a general conclusion in the body of knowledge regarding research studies that have proven that statistically significant gains in software performance testing result when DOE techniques are applied.

Purpose of the Study

Software testing not only affects the future of the businesses producing the software products, but also the businesses using the software as well as members of society in general. The purpose of this research was to evaluate the reported findings from the primary software performance testing studies against the findings from an aggregate of software performance testing studies and add to the current body of knowledge. This research allows an assessment of whether or not measurable improvements in the quality of software testing resulted from applying DOE techniques. This research expanded upon and exploited the sample of isolated studies to assess whether using statistical rigor generalized across a group of software testing studies that applying DOE improves software testing effectiveness and efficiency.

It is important to note that while there was a lack of scholarly literature reporting findings showing positive improvements in software performance testing with DOE across a collective group of software testing studies, there were individual studies that reported positive improvements. Moreover, there were a sufficient number of such individual original studies to perform a meta-analysis to assess the DOE impact on software testing effectiveness on these studies as a group. For example, the rise in flu-like symptoms in a city would be newsworthy for the city to report an outbreak of the flu.

When there are many such reports in many cities and in many states, the news takes on even more import on a national level. Now the newsworthy outbreaks are deemed a flu epidemic. The diagnosis gained strength in the collective body of evidence as presented by all the cities in all the states. Such was the case with this study. With each city's outbreak analogous to an individual study and the outbreaks throughout the country analogous to the studies synthesized in the meta-analysis, this study assessed the impact of DOE on software testing on a collection of software testing studies thus addressed the gap in the scholarly literature.

Nature of the Study

The nature of this study was investigative. The meta-analysis assessed statistically the reported findings from the selected primary software performance testing studies against the findings from an aggregate of software performance testing studies and adds to the current body of knowledge. The selected primary studies were composed of two subgroups, studies that applied experimental design techniques and studies that did not apply experimental design techniques. This research was based on the review of scholarly, quantitative research and subsequent findings for similar software performance testing research investigations. I analyzed statistically the sample population of quantitative research findings, which comprised the data for this research, using a meta-analytic subgroup analysis research method to synthesize and assess, validate, and expand upon the original investigations' findings.

Quantitative study findings take on different forms for meta-analysis. Example forms of interest include difference between group means, correlations between variables,

and proportions of observations (Lipsey & Wilson, 2001). In this quantitative study, I used correlation as the quantitative analysis type to determine if there were an association between two variables of interest. For this research, the analysis assessed the relationship between DOE techniques and software performance testing effectiveness. The meta-analysis synthesized the measure of the strength of the relationship or correlation between the variables of interest from the original included studies. This form of research finding represented covariation for two distinct variables to determine if there was a relationship between them. For this quantitative study, the variables of interest were the dependent and independent variables deemed important factors for improving software performance testing effectiveness.

I selected appropriate effect size statistics to study predictive validity for testing performance improvement from a synthesis of findings across multiple studies. For purposes of this research, the independent input or manipulated variables were selected factors from the primary studies that were deemed to be important in designing test cases. The selected variables of interest were operationalized in numerical format as appropriate for data computation and statistical analysis. The dependent variables were the resulting number of software defects, rate of defect detection, phase detected, and the total testing hours which operationalized the best determination of the software performance testing effectiveness for the selected factors and factor interactions.

Research Questions and Hypotheses

For this quantitative study, the research questions related to improving software performance testing. I discuss the questions and hypotheses in the following paragraphs.

Research Questions

This research focused on the efficiency and effectiveness of software performance testing. This quantitative study addressed the question of whether applying experimental design techniques to software testing improves testing efficiency and software performance testing effectiveness. The key research question for this research was as follows:

- What is the relationship between the DOE techniques (independent variables) applied to test case design and the effectiveness of the software performance testing (dependent variables)?

The investigation of this research question sought to measure, quantitatively, the effectiveness of applying DOE techniques, where software testing effectiveness was defined as follows:

- Improved software quality as measured by more defects found in the overall testing process (i.e., sum total of all defects detected throughout all phases of the software development life cycle).
- Increased test execution efficiency as assessed by the defect detection rate (for example, number of defects detected per hour).
- Improved phase containment of defects, as measured by the number of defects detected in earlier phases in the software development life cycle. This translates into reduced cost, since it is cheaper to fix defects the earlier detected from both software correction and test time perspectives.

- Reduced total number of hours to execute all tests during the software testing process.

Research Hypotheses

Since the research method for this research was meta-analysis, the same hypotheses testing for the variables of interest from the primary studies included were the hypotheses addressed here. I tested five hypotheses for this research.

The first hypothesis centered on the two subgroups that were central to this research. This overall focus was on all of the studies and any of the dependent variables of interests at the subgroup level. Each subgroup included 48 original studies, regardless of the dependent variable of interest.

The focal point of Hypotheses 2 through 5 was a clustering of original studies per dependent variable, as indicated in Table 1. Each of these hypotheses tested the influence of the single dependent variable that was common to all of the included original studies.

Table 1

Hypothesis Makeup by Number of Original Studies

Hypothesis	Dependent Variable	Number of Studies without DOE	Number of Studies with DOE
1	All four dependent variables	48	48
2	Defects detected	20	10
3	Defect detection rate	3	6
4	Defects detected by phase	6	12
5	Total testing hours	19	20

The first hypothesis.

H_{01} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing.

H_{a1} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing.

The second hypothesis.

H_{02} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the sum total of all the valid number of defects detected during the software testing process.

H_{a2} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the sum total of all the valid number of defects detected during the software testing process.

The third hypothesis.

H_{03} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by number of defects detected per hour during the software testing process.

H_{a3} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured number of defects detected per hour during the software testing process.

The fourth hypothesis.

H_{04} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the number of defects detected during the earlier phases of the software testing process.

H_{a4} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the number of defects detected during the earlier phases of the software testing process.

The fifth hypothesis.

H_{05} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the reduction in total number of hours to complete test execution during the software testing process.

H_{a5} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the reduction in total number of hours to complete test execution during the software testing process.

The meta-analytic process for this study included hypothesis testing. It was based on an appropriate, calculated sample size of selected findings from a population of original studies. A test significance level ($\alpha = 5\%$) was used in the hypothesis testing. A standard normal distribution was assumed, and a p value calculated and compared to α . If the p value is less than α , then the null hypothesis is rejected. A more detailed discussion of the hypotheses for this research follows in Chapter 3.

Theoretical Bases for the Study

Prior research related to this area of inquiry traced the application of DOE techniques to software performance testing studies and the review of associated articles by researchers such as Hoskins, Colburn, and Montgomery (2005); Montgomery (2009); Grinder, Offutt, and Nadler (2005); and Kuhn, Wallace, and Gallo (2004). DOE

applications, both classical and the Taguchi approach, by these and other software testing researchers in national research labs, space research, defense-related investigations, and federal agencies projects were evaluated. Factor covering arrays (pairwise and n-way combinational methods) were the DOE techniques utilized in the original studies included in this research.

Sources for the meta-analysis process included Borenstein, Hedges, Higgins, and Rothstein (2009) and Lipsey and Wilson (2001). Both sources described the overall meta-analytic process and addressed some of the criticisms and pitfalls to watch out for. Borenstein et al. examined the statistics of the procedure but from an introductory perspective. Lipsey and Wilson, on the other hand, provided a deeper perspective of meta-analysis from a practical research perspective. Chapter 3 provides a more detailed exploration of meta-analysis.

Definitions of Terms and Acronyms

I operationalized the definitions of technical terms, special words, and acronyms related to software testing, DOE, and meta-analysis used throughout this and the remaining chapters. Following are terms and acronyms used throughout this research study.

Classical DOE techniques are powerful techniques developed by Fisher in the early 1920s at the Rothamsted Agricultural Field Research Station in London for reducing process variation while enhancing process effectiveness and process capability using two-level process parameters or factors (Antony et al., 2003).

Comprehensive Meta-analysis (CMA) software is a computer software package for performing a meta-analysis.

Covering arrays are structures used to represent extremely large input spaces with factors and factor combinations to ensure the maximum testing coverage (Bryce & Colbourn, 2008).

Defects are failures in a software program that are manifested by step or process errors, or an incorrect data definition (IEEE-Std 610.12, 1990).

Dependent variables for this study were defects detected, defect detection rate, phase defect detected, and total testing hours.

Design of experiments (DOE) is a systematic approach to the investigation of a system or process which allows the researcher to organize experiments, collect data, and statistically analyze those data to arrive at objective conclusions (Montgomery, 2009). For software testing, these experimental techniques are applied to significant relevant software testing factors to maximize the number of defects detected using the minimum number of test cases possible.

Effectiveness measures the extent to which desired results are achieved. For this research, it focuses on the generation of the smallest set of test cases whose output results in the detection of the largest set of software defects during the testing process (Gupta & Jalote, 2008; Freedman, 1991).

Effect size is the correlation between variables of interest that provides a standardized indication of the strength of an effect or relationship between the variables (Swanson & Holton, 2005).

Efficiency for software performance testing is measured by the extent to which the desired test results are achieved in the most economical fashion (time and effort) (Gupta & Jalote, 2008).

Fixed effect model is a meta-analytical model having an effect size that is the same for all original studies in the analysis (Borenstein et al., 2007).

Independent variables for this study were the DOE techniques manipulated by the researcher.

Meta-analysis is a process composed of statistical research methods and techniques of quantitative research synthesis that focuses on the aggregation and comparison of conceptually comparable studies with similar statistical form (Lipsey & Wilson, 2001).

Random effect model is a meta-analytical model in which the effect size represents an estimate of the mean of a distribution of effects from the effect sizes from each of the different included participant studies (Borenstein, 2007).

Software development life cycle is the period that begins with the decision to develop software or a software product and continues through to its delivery (IEEE-Std 610.12, 1990).

Software development process is the process in which a customer's requirements are translated into a software product. The process typically involves gathering user needs and translating them into software requirements, transforming the requirement into the software design, coding the design, and finally testing the code to ensure it meets with

the customer's needs. These activities are not necessarily linear but may overlap and are often iterative (IEEE-Std 610.12, 1990).

Software performance testing is an activity in which a software program or software system is executed under specified conditions and the results are evaluated in order to verify the quality (IEEE-Std 610.12, 1990).

Software product is the finished, complete software set composed of computer software programs, procedures, and the associated documentation and data or any one of these items (IEEE-Std 610.12, 1990).

Software quality is the quality of a software program or system refers to the degree to which its design and development meet specified requirements and produce the desired results.

Software tester is the person who conducts the software test suite or test cases execution task(s).

Taguchi DOE approach was developed in the 1950s by a Japanese quality engineering Guru, Dr. G. Taguchi, and was introduced into the United States in the early 1980s. This approach is based on a single large experiment where all the main effects and some important interactions are studied. It also uses linear graphs for assigning various factors for processes or parameters of three or more levels (Antony et al., 2003).

Test case is a set of input values, pre-execution conditions, specified execution conditions, expected results, and post-execution conditions, whose objective is to exercise a software program or to verify compliance with a specific requirement and desired effect (IEEE-Std 610.12, 1990).

Assumptions, Limitations, Scope, and Delimitations

The scope of this research was delimited by the quantitative findings of the primary studies included in this investigation.

Assumptions

The main assumption was that all of the researchers in the primary studies that applied DOE techniques were proficient in experimental design techniques and that the testing activities were conducted by experienced software testers. By their reporting in scholarly peer-reviewed literature, it was presumed that these studies represented valid research by competent researchers. Since these assumptions were not verified, they also pose potential limitations for this research.

Limitations

The statistical generalizability of this research is limited by the original software testing research studies composing the sample population. From this sample population of included original studies, generalizations for all software testing studies are concluded. An additional limitation was posed by the fact that the software testing environments, software implementation languages, and types of software systems varied. The generalizable concept as related to this research is explored in detail in the meta-analytic research methodology discussion in Chapter 3.

Scope

In spite of the many technological advances and the proliferation of software test tools, the fact remains that it is impossible to be certain that a software package or a software-based system will function flawlessly. Ironically, because of technological

advances, society's level of expectation for safe software systems and software products has been heightened, and flawless functioning software is exactly the expectation. As software and software-controlled systems become more complex and thoroughly entwined in the everyday framework of our society, their potential for costly and even life-threatening failures continues to grow. The scope for this research was the same as those of the included recent primary studies covering the same software factors deemed important in effective software performance testing. It addressed software quality as measured by the effectiveness of the testing performed to decrease, to the extent possible, based on the number defects detected and the software development phase in which the defects are detected

Delimitations

This research study was bounded by the range of the investigations of the key questions of the included primary research efforts. An ultimate resolution or solution for software performance testing issues was not suggested, and neither was software testing reliability, the number of defects remaining in the software product after customer delivery, the mean time before failure, or the mean time between failures.

Significance of the Study

The most observable activity of software testing is the test case execution. However, to be effective and efficient, Iacob and Constantinescu (2008) asserted that the testing phase activities for planning the testing, designing the test cases, preparing for the test execution, and evaluating the test status should be equally visible. Since this research validated the assertion that incorporating design of experiments techniques in the

planning and test case design and test case selection improves the effectiveness of software performance testing, then perhaps more software test organizations in private industry will adopt this testing methodology in their software development process.

Gap in Current Literature

Currently, the literature reports single-event investigations for a particular software testing effort. This research addressed the gap in the current scholarly research literature of findings from software performance testing investigations which encompassed an assessment from a group perspective presenting an understanding derived across multiple research studies where each included primary study conceptually addressed the same software testing performance issues. The significance of this research was to see how the findings of individual studies compare to the results from the aggregate of all the studies in terms of the quantitative improvements in efficiency, effectiveness, and cost in software testing exercises which incorporated experimental design techniques. Testing improvements, for the purpose of this research, were measured by the total number of defects identified, with emphasis on the defect detection rate (especially those detected in the early phases in the software development life cycle), as well as the test execution time as it relates to testing cost.

Professional Application

The potential of this research for the software profession is as an approach to the software testing phase that elevates it to the same technical level with the front end phases of the software development process. Rather than an approach based on the whims of the software testers or a group of testers whose main goal is to break the system under

test, this research offers an alternative scientific approach for applying technical expertise to the software testing phase of the software development life cycle. Findings could boost the credibility of software testing, of the software testers, and ultimately the software testing profession.

Implications for Positive Social Change

The implications of this research are not confined to software development organization or the software testing profession. There are also potential positive implications for society in general. Software products are so commonplace that everyone in society is affected by the perception the products they are using are safer. For mission-critical software, especially that which is used in products that support U.S. men and women in the armed services, better software performance testing provides a certain amount of assurance that engenders their confidence in the software and software products they use. Additionally, as the testing costs are reduced, the total cost for producing software products is lessened. The software end-users stand to profit from this cost reduction in the form of less expensive software products. Society then is able to enjoy more affordable software products upon which they have become increasingly dependent for purposes such as educational, medical, career, and entertainment, to name a few.

Summary

This chapter identified the gap in the current literature on quantifiable measures of improvements across multiple studies that have investigated the effectiveness of software performance testing for producing quality software. The theoretical bases for this

research are factors covering experimental design techniques and the meta-analysis research design. The purpose of this research was to assess the reported findings from the primary software performance testing studies against the findings from an aggregate of software performance testing studies. While addressing the current state of software testing, this study also added to the software testing body of scholarly knowledge by showing whether measurable improvements in the quality of software testing results from applying DOE techniques. The next chapter contains a review of the primary research studies from the peer-reviewed research literature and integrates the corresponding findings included in the analysis. Chapter 3 includes the methodology, Chapter 4 contains the results, and the conclusions and recommendations are contained in Chapter 5.

Chapter 2: Literature Review

Introduction

The focus of this chapter is the search strategy and review of primary research studies from the peer-reviewed literature on the effectiveness of software performance testing. This emphasis on the effectiveness of software testing was the common theme, and the main research criterion for the inclusion in this study was that it was peer-reviewed, recently published literature. The problem addressed in this research was the gap in the scholarly research literature of concerted, concentrated efforts that have assessed software testing performance improvements with objective statistical data across a group of selected studies. The aim of this research was to assess the reported findings from the primary software performance testing studies against the findings across a group of such software performance testing studies.

This chapter begins with a discussion of the general search criteria, strategy, and techniques used to research the literature for this study. The structure of this chapter follows a format progressing from a general discussion on the theoretical basis for the study and the research design to a specific focus on literature supporting the topic for this research. From a discussion of general theoretical concepts on software testing, experimental designs, and meta-analysis, the discussion flows to the relationship and effect of applying DOE techniques to software testing to the resulting software performance testing effectiveness. Once this framework for the review has been established, the emphasis of the discussion turns to the actual literature reviewed for this research study. This critical examination covered relevant quantitative findings from

original research studies that assessed performance improvements from applying experimental design to software testing.

The exploration of literature in this chapter presents a critical comparison, contrast, and examination of the research findings reviewed. Specifically, this review describes what research was conducted already in this area and went further to show that the research conducted, with statistically significant results, seemingly was performed as isolated efforts. The lack of concerted, cohesive software testing research efforts has done little to validate these findings or address them in a manner to make any significant impact on industry testing practices or societal concerns with the effectiveness of software performance testing and the quality of software products. Finally, this chapter concludes with an examination of the reviewed literature from the perspective of the variables of interest germane to this study.

Literature Search

This section includes a discussion of the search strategy employed for finding prior research studies related to the general theoretical concepts (software testing and experimental design techniques) of this study and explains the review criteria used for including the original studies in this meta-analysis. I then examined and discussed the chosen original research in light of the gap that was the focus of this study.

Search Strategy

Once I defined the theoretical concepts and themes for this study, I devised a strategy for searching the peer-reviewed literature. For all types of literature (i.e.,

academic journals, technical journals, and technical reports), the strategy included the following:

- searching online databases,
- creating keyword lists,
- using bibliographies for references, and
- using renowned researchers in the areas of interest as leads.

Online databases. A typical first approach for researchers is to search common online databases using relevant keywords (Timulak, 2009). Searching online databases was how the research began. A significant advantage for using online databases is the feature to only search peer-reviewed literature. That was a major criterion for selecting literature with reports of the original research to be included in this research study. From the database searches, only peer-reviewed journals and reports were reviewed for fit and relevance.

Keywords. Examples of keywords and key phrases in my literature search are *testing, software testing, test design methodology, experimental design techniques, and DOE techniques*. Using a relevant keyword literature search strategy allowed for being mindful of quality and study validity of the original research and for weeding out those studies that merely created new questions.

Bibliographies. For those initial online database searches producing articles or reports that proved promising, I used the bibliographies in those studies to create a list of references and contributors who had also done research in the relevant areas. This list provided leads for additional literature searches.

Recognized researchers. The online database searches were also useful for revealing others who had done research in the areas of interest for the themes for this research. The names for these researchers proved to be useful for *author* keyword searches and for finding their works. For example, the NIST scientists, Kuhn, Kacher, and Lei (2008) have done extensive investigations in software testing with experimental design techniques, and Montgomery (2009) is an experimental design techniques expert.

General Theoretical Concepts

Theories on software testing studies and experimental design techniques are cited in the research literature ranging from technical journals to reports from scientific studies conducted by government-sponsored organizations to academic research publications. Detailed discussions follow on software testing strategies, DOE techniques, and meta-analysis.

Software Testing

There is plenty of research being conducted in the software testing domain where experimental design techniques were used. This is evidenced by the number of original research investigations reported in the literature and those listed in the references section of this paper for inclusion in this research. For example, Sjoberg et al. (2005) published a survey of such engineering test pursuits. Lazic and Velasevic (2004) equated software testing effectiveness with the percentage of defects detected and the defect containment, while software efficiency was measured by the dollars and hours spent per detected defect. However, there was a gap in the scholarly research literature of concerted, concentrated efforts among such researchers quantitatively to validate measurable

software testing performance improvements with statistically significant data across a group of studies with comparable research designs.

The software testing phase is to ensure that the developed software product meets customer requirements, is free of software defects (to the extent possible), is ready for customer delivery, and is safe for the customer's use. In other words, the delivery of a quality software product is the ultimate goal. Early on, software testing researchers like Iacob and Constantinescu (2008) supported spending more time in the software development life cycle with up-front requirements analysis and design phases to lessen the time spent in the software testing phase, which tends to take up more than its share of the software development time. The software testing community has since found from recent research that the emphasis should be equally divided between upfront planning and analysis focused on the test phase to address the associated time and cost incurred during testing. Bryce and Colbourn (2006) would probably have concurred as they reported, based on NIST data, that more than \$60 billion a year was spent on software defects due to expenses for test execution costs associated with software testing alone.

A common testing misperception, according to Iacob and Constantinescu (2008) is that software testing is just running test cases or running the software programs. The reality is that the actual test execution is only part of testing phase of software development life cycle. Testing activities begin before executing test cases and continue even after the software testing is completed. Testing activities include test planning, selecting test conditions, designing and selecting test cases, determining expected results, assessing test results, evaluating the testing effort completion criteria, test status

reporting, and finalizing or closure of the test phase. Software testing activities also cover the creation and review of test documentation upfront, as well as static analysis of the test results.

In recent years, researchers have come to understand that effective software testing is best achieved by using a structured and scientific methodology, rather than the historical *break-it* mentality (Iacob & Constantinescu, 2008). The goal of the software testing effort is for maximized effectiveness through the design and development of a more technical testing strategy, sound test methodologies and practices, and the use of software test tools and techniques. The total software testing performance effect, ideally, would be the delivery of a quality software product with corresponding lower cost. Such an occurrence would not only be beneficial to the software development life cycle costs but ultimately to the business. Since the main test execution process is one of the last software development life cycle stages, it must be both thorough and efficient in order to maximize effectiveness and add quality to the testing process. Historically, software testers did not have to have specialized knowledge in order to break the software system as noted by Iacob and Constantinescu. In the current testing industry, however, the prevailing thought seems to be that in order to have quality testing processes in place, the tester needs a deep level of understanding for how the software actually works.

Coupling various coverage-based software testing criteria with an experimental design technique has proven most viable in addressing software testing effectiveness and efficiency when comparing defect detection abilities. Examples of code-based testing criteria include block coverage, branch coverage, and predicate coverage. Coverage-

based testing assumes the researcher or experimenter has some knowledge of the software under test. Coverage criteria may be achieved by using covering arrays. Covering arrays are structures for representing extremely large input spaces often used to efficiently implement black box software testing. Additionally, technical proficiency and mathematical expertise have recently proven to be beneficial skills. Testers have developed several algorithms for generating software testing covering arrays (Cohen, Dalal, Fredman, & Patton, 1997). There are two rival aims for these algorithms, which are to minimize the time required to produce the test array and to minimize the number of rows in the test array (Forbes, Lawrence, Lei, Kacker, & Kuhn, 2008). In the case of either goal, reducing either the execution time or the resulting covering array size, there are potential improvements for software testing performance from a cost perspective. Pairwise testing and combinatorial testing are two such covering array strategies requiring technical expertise beyond software coding.

Pairwise testing. To illustrate the use of pairwise testing coverage in the reduction of the test suite size for software testing, Cohen et al. (1997) explored the greedy algorithm. This illustration assumed a structure with t test factors where the i th parameter has l_i different values. It further assumed that r test cases have already been selected. The $r + 1$ test case was selected by first creating M different potential test cases and then selecting the test case with the best coverage. Each potential test case is selected as follows:

1. Select a factor f along with l for f so that the selected factor shows up most often.

2. Let $f_j = f$. Then randomly select all the remaining factors for the t test factors $f_1 \dots f_t$.
3. Let the value chosen for f_i be labeled v_i , where $1 \leq i \leq j$. For every f_{j+1} , select a value v_{j+1} , where v_{j+1} is one of the occurrences to appear most often in the resulting pairs.

In Step 3, each parameter value is considered only once for inclusion in a potential test case candidate. Also, when choosing the value for parameter f_{j+1} , the possible values are compared with only the j values that had already been selected for parameters $f_1 \dots f_j$. This algorithm can potentially reduce the number of generated test cases by 10% to 20%, resulting in smaller test suites. This trait was an important consideration for its inclusion in this research. Smaller test suites that cover more software functionality and complete execution in less time translate to reduced total testing time. Reduced test execution time accompanied by greater test functionality coverage is an indication of a measurable improvement in the testing activity. In the vein of continuous improvement efforts, reducing test suite sizes and reducing test execution time continue to be an ongoing research area of interest.

Combinatorial testing strategy. For especially large software products, where complete or exhaustive testing is impractical, if not impossible, combinatorial testing techniques are very effective at uncovering software defects. This testing strategy based on combinatorial design is used to generate test cases that cover pairwise, triple, or n -way combinations of a system's test parameters. Test cases are developed for each different combination of parameter values. For n -way combinations, the test cases for a fixed n

may grow logarithmically. According to Bryce and Colbourn (2007), such computational techniques can take a covering array through a number of transformations, first computing the cost of the change then accepting the change based on a predefined interaction acceptance test criterion. A common example for this testing strategy is called combinatorial interaction testing (CIT).

A CIT testing strategy involves a mathematical construct called a covering array. Covering arrays are arrays derived from a set of symbols having the property that every generated subarray includes these same symbols at least once (Cohen, Dwyer, & Shi, 2008). The *in parameter order* (IPO) greedy algorithm is an often-implemented CIT testing for n -way test coverage with large software products. The IPO algorithm generates n -sets for the first n factors and then incrementally expands the solution, both horizontally and vertically, until the solution is complete. By definition, combinatorial testing includes pairwise testing, but for purposes of this research study, combinatorial testing referred to n -way testing where n represents more than two test factors. Kuhn, Kacker, Lei, and Hunter (2009) recommended combinatorial testing as an efficient method for detecting hard-to-find software defects.

DOE Strategies

All research involves definite procedures or sets of procedures, observers, and experimenters. Research efforts confined to one experiment or test limit the validity of the findings. Replication is crucial to experimental design as it permits the researcher to address external validity and increase the generalizability of theories and hypotheses (Singleton & Straits, 2010). According to Montgomery (2009), it is important to

approach experimental design statistically because of the objective nature of statistical approaches. Montgomery further theorized that the application of experimental design techniques early in any process could substantially reduce cost and result in processes and products that perform better. Each of the original studies selected for inclusion in this research met the inclusion criteria of being a software testing investigation based on testing involving the interactions of two or more test factors. Montgomery defined factorial experimental designs as complete replications of all possible combinations of the levels of the factors investigated. For example, if there are a levels for factor x_1 , b levels of factor x_2 , c levels of factor x_3 , and so on, then there are $a \cdot b \cdot c \cdot \dots \cdot n$ total combinations. The factor interactions terms become independent variables and are represented by $x_1x_2, x_1x_3 \dots \cdot x_1x_n$ and so on. Because of the iterative nature of experimental designs, these combinations of variables and interactions are repetitive for each combination in the course of an experiment. As these replications are investigated so are the interactions between these factors. For this quantitative study on software testing, the statistical analysis of the experiments for two factors and the interaction between the factors, the two way interactions were represented in an analysis of variance (ANOVA) regression model representation written as,

$$y = \beta_0 + \beta_1X_1 + \beta_2X_2 + \beta_{12}X_1X_2 + \epsilon, \quad (1)$$

where y is the response, the β s are the parameter values to be determined, X_1 is a variable representing the first factor, X_2 represents the second factor, and ϵ denotes the random error. From the analysis, statistical data provide insight into how each factor and factor interaction impacts the resulting testing performance.

Experimental design techniques can be broken down into two major categories, *classical* and *Taguchi*. Those DOE techniques that are typically applied in hypothetical testing investigating potential parameter interactions are referred to as classical techniques. Researchers who prefer the Taguchi DOE approach generally prefer applying three or more levels of the process or design parameters to estimate nonlinear effects. The Taguchi approach is often thought of as parameter design according to Antony (2006). He, Staples, Ross, Court, and Hazzard (1997), however, submitted that this method is much more. It is system design, parameter design, and tolerance design, where the constraints provide for tolerance in case the desired results are not realized during system and parameter design. An additional distinction between the two approaches proposed by Antony is that classical DOE strategies support a sequential and adaptive approach to experimental design, whereas Taguchi's approach typically exploits a single large experiment to study all possible main effects.

Classical DOE techniques. Classical DOE was created by Fisher in the early 1920s. Fisher and his coworkers made major breakthroughs in design and analysis of experiments and were among the primary contributors to the literature early on. Montgomery (2009) emerged as an expert on classical design and analysis of experiments in recent years. Researchers utilizing classical DOE techniques are driven by the fact that this approach permits the investigation of process factors for at least two levels so that critical process or design parameters can be identified early in an investigation.

DOE techniques typically focus on various coverage criteria. This research, centered on the application of the design and analysis of experiments to software testing,

exploited this coverage criteria feature. The coverage criteria played a key role in experimental designs that assume that a model under test and test cases or test suites can be systematically generated by covering certain aspects of the experimental model (Gupta & Jalote, 2008).

Taguchi DOE approach. Taguchi, a Japanese quality engineering guru, developed the Taguchi approach to DOE in the 1950s, and it was introduced to the United States in the early 1980s. Experimental designs following this approach are typically concerned with the optimization of a single quality trait or response. Taguchi's approach to DOE concentrates on the robustness in the functional performance of a process or design. The researcher's goal in the Taguchi approach is identifying the best level for a given process or design according to Antony (2006).

Further, He et al. (1997) described traits for the Taguchi approach that make it perfectly suitable to software testing improvement research and for this research. These traits for Taguchi's approach to DOE are summarized as follows:

- Viewing processes as transformations and quality engineering as a transformation optimization method.
- Defining product quality by the least amount of loss in the functionality after the product delivery.
- Developing as an engineering experimentation technique.
- Designing to minimize the number and iteration of experiments.
- Helping engineers improve products and processes was a main goal.

Choosing the appropriate DOE strategy. Even with the clear division of camps for DOE techniques that researchers use, there is still an issue among researchers as to which DOE strategy to follow and when to employ the strategy. The choice of the appropriate DOE strategy, classical or Taguchi, depends on a number of factors. Such factors include the nature of the problem, the degree of optimization sought, time and costs constraints, the amount training needed on the DOE approach, and statistical validity and robustness desired. Antony (2006) proposed a simple strategy selection framework to address this issue of which DOE strategy to follow. This framework, presented in Table 2, has been validated by a number of DOE researchers.

Table 2

A Framework for Choosing the Appropriate DOE Strategy

Nature of the problem	Taguchi DOE	Classical DOE
Experiments with strong interactions are anticipated by the experimenter	x	√
Rapid process understanding and quick response to management	√	x
To determine the optimal condition of the process	x	√
To achieve robustness and noise factors are identified as a source of variation in the process	√	x
To predict a target value for the process performance characteristic	x	√
Reduce variability around a specified target value and quantify the loss associated with it	√	x
To develop a mathematical model connecting the response (output) and a set of process parameters and their interactions	x	√
To set tolerances on the critical process/design parameters for achieving desired variability	√	x

Note. x – not recommended; √ - recommended

Note. From “Taguchi or classical design of experiments: a perspective from a practitioner,” by J. Antony, 2006, *Emerald Sensor Review*, 26(3), p. 228. Reprinted with permission.

From the framework presented in Table 2, depending on the specific nature of the problem, the implementation of either DOE strategy seemingly would lead to improvements in a process or a resulting product. For this quantitative study, the experimental design techniques focused on the evaluation of software performance testing effectiveness and testing efficiency. On an initial review of this table, the classical DOE strategy appears to be the more appropriate strategy for this research study. This observation is supported by the listed recommendations presented, such as mathematical modeling, experiments observing strong interaction, and the search for optimal conditions for a process, which is the software testing process in this instance.

Meta-Analysis

Meta-analysis refers to the statistical synthesis of findings from a series of empirical research studies (usually quantitative), where each original study deals with the same constructs, the same relationships, and similar findings are represented in the same statistical form (Borenstein et al., 2009; Lipsey & Wilson, 2001; Singleton & Straits, 2010). Meta-analysis is applied in many fields of research for various purposes, including validating findings from original studies, synthesizing available research data in order to set policy, and directing new research. Meta-analytic procedures provide a systematic analysis of findings from literature reports of prior quantitative studies for the purpose of integrating the findings. Many meta-analytic studies are performed to assess the reliability and generality of findings from prior research studies, to test new hypotheses, or to assess the relationship between an explanatory variable and a response variable.

A meta-analysis typically has one of three main goals: (a) to test whether the results for the included studies are homogeneous or the mean effect size represents the effects of all studies in a group, (b) to find an overall index for the effect size of the observed relationship, for a specified confidence interval, or (c) to determine if there is heterogeneity (variability in effect sizes) among studies (Fitzgerald & Rumrill, 2003; Lipsey & Wilson, 2001).

Once the problem statement and the research questions have been well defined with the variables/relations of interest identified, the researcher follows the following basic steps to perform the meta-analytic procedure:

- examine and review the literature to collect studies with findings of interest,
- code selected studies to format the variables of interests and study characteristics into variables with measurable units,
- calculate effect sizes and data computations,
- analyze the data and interpret the results, and
- report the findings.

During the meta-analysis, these steps are performed to estimate the overall strength of the relationship between the independent and dependent variables of interest as well as the effects of any other variables from the included studies (DeCoster, 2009). Chapter 3 presents a more detailed discussion of the steps for the meta-analysis research method.

Lipsey and Wilson (2001) compared meta-analysis to survey research in the sense that for meta-analysis research literature is surveyed, whereas in survey research people are surveyed. It facilitates a systematic review of the peer-reviewed research literature.

While this alone offered a compelling argument in support of meta-analysis, the key characteristic for this research design was the statistical standardization provided. This standardization is in the form of an effect size statistic which defines the quantitative findings of a set of research studies permitting meaningful statistical comparisons and analysis. This feature of the research methodology fits well with the goal for this research effort. Moreover, this inherent nature of the structured meta-analysis research methodology supports the research process, in general, and specifically in terms of the methodology's literature search strategy and criteria for selecting the original studies.

Although several different types of meta-analysis are used in the social sciences (for example, the cluster approach, the validity generalization approach, the "Glassian" approach), the approach advanced by Glass appears to be the most commonly utilized in the social science literature (Fitzgerald & Rumrill, 2003). A meta-analysis uses statistical techniques for combining data from primary studies into a weighted pooled estimate of effect sizes. The resulting weighted estimate is a summary estimate with a 95% confidence interval. There are three different methods used for combining the data, which are random effects, fixed effects, and mixed effects. The random effects and fixed effects frameworks are typically used more often. In both frameworks, the pooled weighted average is calculated from the statistical findings of the primary studies. In the fixed effect models, the data between the primary studies is assumed to be the same with any differences assumed to be because of random error. For random effect models, there is typically heterogeneity among the primary studies and the resulting weighted estimate is often more conservative than with fixed effect methods (Turlik, 2010). Singleton and

Straits (2010) declared that the main distinction between the random effects and fixed effects frameworks is that fixed-effects frameworks treat between-study variability as derivatives of sampling and other chance processes while the random effect model attributes such variability to differences in methods, conditions, and research settings.

Effect size, as defined by Coe (2002), is a tool for quantifying, reporting, and interpreting the size and effectiveness of the difference between two groups. Coe further suggested that the strength of this tool is that it allows the researcher to move beyond the question of whether a variable or factor makes a difference to the far more meaningful question of how much of a difference the factor makes. According to Borenstein (2009), since it is often not possible to know the effect size for the original primary studies, the effect size is often estimated by the researcher. Moreover, Coe considered that placing the emphasis on the size of the effect (a measure of the significance of the difference) in a research effort promotes a more scientific approach to the accumulation and synthesis studies for adding to a body of knowledge in any research domain. An effect size is equivalent to a *Z score* (standardized score) of a standard normal distribution. Another interpretation of effect sizes is that they make use of equivalence between the standardized mean difference, d , and the correlation, r . Still another interpretation for effect sizes is as a comparison of them to other effect sizes of differences that are already known. A noted advantage of using effect sizes in research is that after an experiment has been replicated multiple times, the different effect size estimates can be combined and synthesis using meta-analysis to give an overall best estimate of a measured effect for the research.

Coe (2002) submitted that confidence in an estimated effect size can be increased if the statistical significance, which is usually calculated in the form of the p value (the probability that the difference of at least same size would result even if there were no difference in the original studies) is incorporated. This p value is typically calculated from a t test (a paired-observation comparison test). If the p value is less than 0.05, the effect size is generally considered large enough to be significant. In estimating effect size, Coe cautioned that it is most important that the margin of error is also reported. The margin of error is calculated using the same data contained in a significant test based on the concept of a confidence interval. A 95% confidence interval is equivalent to a 5% significant level. Borenstein et al. (2009) found that p values are often misinterpreted and should never be used in place of effect sizes. For example, a p value labeled significant could reflect a large effect size or it could also reflect a small effect size measured for a large study. Conversely, while a nonsignificant p value might suggest a small effect size, it could also reflect a large effect size measured for a small study. Hence, reporting the margin of error when p values are used is one way to prevent misinterpretations. Further, to avoid any potential for misinterpretation, Borenstein et al. advised working solely with effect sizes directly rather than just p values. In conducting meta-analyses, where the goal is to synthesis findings from multiple original studies, these researchers warned that the use of effect sizes is crucial in the research process.

Calculating effect sizes and corresponding variances is relatively straightforward if the summary data such as the mean, standard deviation, and sample size for the original studies are available. In practice, however, it is often not possible to have full access to

such data, and neither is the effect size for the original primary studies known, so the effect size is often estimated. Three major considerations that should drive the choice of the effect size index according to Borenstein et al. (2009, p. 18) are:

1. The effect sizes should be comparable in that they measure the same thing.
2. The estimate of the effect sizes should not require a re-analysis of the data.
3. The effect sizes should have defined technical properties so that the sampling distribution should be known so that variances and confidence intervals can be computed.

Meta-analysis is more than simply producing a combined effect size. The primary studies are examined and analyzed for differences and an understanding of what factors drive those differences. Revealing characteristics and relationships between effect sizes in the context and design of the original studies are also important goals in conducting meta-analyses. Results from a meta-analysis across multiple original research studies tend to be highly statistically significant, thus increasing confidence in their generalizability (Coe, 2002). In addition to statistical significance, DeCoster (2009) insisted that the true value in performing meta-analyses is found in the theoretical interpretation and integration of findings showing how the original included studies are consistent or inconsistent in the issues studied. The original studies were empirical in nature, examined the same constructs and relationships, and had quantitative findings presented in comparable statistical format.

In meta-analysis, the unit of analysis is each individual primary study. The meta-analytic data analysis usually begins by defining the distribution for the set of effect sizes

for each of the primary studies, according to Lipsey and Wilson (2001). Then using tools like breakout table, ANOVA comparisons or multiple regression, the researcher examines the relationships between effect sizes and the study variables of interests. Before getting in to the heart of the statistical analysis, it is often necessary for the researcher to adjust the individual effect sizes from the primary studies for any bias. There are three most commonly used effect size statistics for correcting bias. These are the standardized mean difference, the correlation, and the odds ratio. The standardized mean difference is the index created by dividing the raw mean difference for each original study by its standard deviation. The correlation coefficient measures the linear relationship between two variables of interest. In meta-analysis, the correlation coefficient may function as the effect size index. Odds ratio is the ratio of two odds. This effect size statistic compares two groups in terms of the odds of an event or status occurrence and is applicable to research findings that use binary data, according to Lipsey and Wilson. The correlation coefficient and the odds-ratio are translated into formats more convenient for the actual statistical analysis and then converted back to their original format for reporting the meta-analysis results (Borenstein et al., 2009).

Sample size considerations. In research, the question of sample size is always one that has to be addressed. Ideally, the entire target population is the best because then there is less uncertainty to deal with. In practice, this is often impractical due to either budget constraints, time constraints, or some combination of the two. According to Timulak (2009), the key criterion for literature inclusion is whether the original study under review addressed the research questions for the meta-analytic research study. For

meta-analytic studies, as few as two studies may be included but at least a dozen is typical, as suggested by researchers Dieckmann, Malle, and Bodner (2009). Other researchers, such as Singleton and Straits (2010), felt that certain interrelated factors should be part of mathematical equations for calculating the sample size for a research study. The five factors are (a) the heterogeneity of the target research population, (b) the researcher's desired resulting precision, (c) choice of sampling design, (d) available time and financial resources, and (e) complexity of planned data analysis. The quality and accuracy of research efforts and resulting findings can be directly linked to the appropriateness and adequacy of the sample sizes used.

Sample size determination is an integral step. In the original studies included in a meta-analysis, Borenstein et al. (2009) hypothesized that power analysis (analysis of the likelihood of a test giving a statistically significant result) was vital to the sample size determination. The power analysis is very similar for meta-analysis as for the original studies. For meta-analyses the statistical significance is strongly linked to the effect size. Rather than the dependency on sample size, the power in meta-analysis is dependent on the inclusion criteria as the sampling design for the choice of original studies to be included in the study. The reason for this stems from the fact that even for large samples in a meta-analysis, if the methodologies vary from study to study or the findings are inconsistent, then the validity of the research could be in question. Conversely, a meta-analysis for a few select studies with carefully chosen inclusion criteria such that the same methodology was used and the findings were consistent from study to study could result in a precision study of high validity.

The size of the overall target population is unknown, which makes it difficult to know the shape of the studies population distribution. According to Aczel and Sounderpandian (2006), when this is the case the rule of thumb for sample size determination is that a sample of 30 or more elements is considered large enough for the application of the central limit theorem. For a sample size selection following this rule of thumb, the sampling distribution of X-bar is normal, the expected value of X-bar is μ (mean), and the standard deviation of X-bar is σ/\sqrt{n} .

As far as a maximum number of articles or literature to include in a study of this type, these authors recommended that the number not exceed 100 primary research studies. Hence, I had to be content with determining a minimum sample size to satisfy some set precision requirements. In order to calculate the minimum required sample size for a research study, Aczel and Sounderpandian (2006) advised that the researcher answer the following three questions:

1. How close should the sample size be to the true, unknown value?
2. What should the confidence level be so that the distance between sample size and the unknown parameter is less than or equal to the answer for question number one?
3. What is the standard deviation for the target population for the research effort?

Depending upon the answers to these questions, the answers can be plugged into a formula to calculate the minimum required sample size. For example,

$$n = \frac{Z^2\sigma^2}{c^2}, \quad (2)$$

where n is the minimum sample size, Z is a standard value which is 1.96 for a 95% confidence level, and σ is the sample standard deviation derived using the formula,

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}, \quad (3)$$

where \bar{x} is the sample mean, x_i data points (where $I = 1, 2, \dots, n$), and c is the confidence interval. While calculating the required sample size, the researcher is also trying to minimize the chances for errors. In statistical hypothesis testing, rejecting a null hypothesis is called a Type I error. Failing to reject the null hypothesis is called a Type II error.

If both types of error are costly, the typical action for the researcher is to increase the sample size to ensure greater validity in the research results as suggested by Aczel and Sounderpandian (2006). There are instances, however, where this is not practical as in meta-analysis. In meta-analysis, the alternative to requiring a minimum sample size is to increase the reliability of the selected sample included in the study. According to Borenstein et al. (2009), the focus for meta-analysis is the inclusion of samples that meet predefined criteria and balancing the Type I and Type II errors, rather than sample size calculation. As for the number of original studies to include, the research discipline or domain seems to be a major consideration. Effect size calculations, on the other hand, are much more critical to the validity of the meta-analysis results. Specific sample size considerations and calculations are discussed in detail in Chapter 3. In addition to addressing the research questions, the included studies also focused on the significant

impact, clearly detail processes and steps that led to the significant impact, and examine the theory or methodological framework of this research.

Study inclusion criteria. In meta-analysis, after a research topic has been chosen, the meta-analyst very specifically and very clearly defines the research domain for the literature of primary studies to include. Similarly for this research study, identifying the gap as a lack of scholarly research literature reporting statistically-significant research results validating measurable software testing performance improvements across a group of research studies was significant as key to identifying the relevant research domain. This domain was appropriate for this research study and for including in a meta-analysis. The research population consisted of the peer-reviewed literature reporting findings on the correlation between software performance testing results and testing practices where experimental design techniques were applied. In meta-analysis, Lipsey and Wilson (2001, pp. 17–18) suggested developing general categories for primary studies inclusion criteria. With their recommendations in mind, the primary studies inclusion criteria used in this research study were as follows:

- The study investigated the effectiveness of software performance testing.
- The study included the application of experimental design techniques.
- The common key variables of interests for the original studies were the variables interest for this research.
- The study utilized a quantitative research design.
- The study was reported in a peer-reviewed technical journal or academic publication.

- For the studies that did not apply experimental design techniques, the study findings showed conclusive software performance testing improvement.

Literature Examination and Analysis

This section is a discussion of the research literature reviewed for this research. The literature review describes the use of testing methodologies, like pairwise and combinatorial test strategies, in software testing investigations. The different research findings and researchers points of view are compared and contrasted. Although the focus of the literature and research was recently published research studies, during the search there were studies conducted early on that had direct relevance and implications for many of the newer more recent studies. Several of these early studies discussed here showed the relationship between earlier works and the literature reviewed and included in this research.

Early Prior Research

Early research in this area included Dalal and Mallows (1998), who researched software defect reduction using two-factor or pairwise test covering techniques. Researchers Dunietz, Ehrlich, Szablak, Mallows, and Iannino (1997) studied software testing with emphasis on defects and execution times. Cohen et al. (1997) were also among the early experimenters to investigate test improvements from test suites sizes based on combinatorial interaction covering techniques. These earlier researchers had in common the fact that they used experimental design strategies. On the other hand, He et al. (1997) were successful in efforts showing performance testing improvements employing the Taguchi approach. These researchers were pioneers and their efforts paved

the way for the later research into applying experimental design to the software testing. These studies investigated pairwise and combinatorial testing strategies. These researchers paved the way, encouraging further study of ways to improve software testing effectiveness. Some of these subsequent research efforts were the focus of this literature review, showing the history of this problem area and the continued efforts to improve software testing effectiveness.

Current Research

The research findings reviewed covered studies on software performance testing, experimental design in software testing, meta-analytic process, and the studies to be included in the meta-analysis procedure for this study. The following discussion examines the findings and presents a comparative analysis of the reviewed research studies with regard to the common theme of this research, which is assessing whether the impact of applying experimental design techniques to the software testing process improves the effectiveness of the software performance testing.

Experimental design techniques are essentially testing techniques. The experimenter develops the hypothesis then proceeds to set up an experiment to test said hypothesis. The setup and test activities are iterative until the researcher is convinced that an adequate number of experiments (tests) have been performed to objectively show that an observed cause-effect relationship or pattern exists. Seemingly, it is fitting to apply experimental design techniques to any situation where tests were performed to observe if there is a correlation between an action and some effect. Examples of researchers who did just that include Kuhn, Wallace, and Gallo (2004) of the National

Institute of Standards and Technology (NIST) for software testing; Montgomery (2009), an expert in experimental design techniques; Borenstein et al. (2009), experts in meta-analysis; and Fitzgerald and Rumrill (2003), who are also meta-analysis experts.

Targeting test cases and test suite size reduction, others researchers investigated ways to improve software testing performance. Literature with this as the common research theme that applied pairwise software testing DOE strategies included Tai and Lie (2002) who conducted research involving a pairwise testing strategy. Bandurek (2005) investigated test cases selection by applying classical DOE strategies using covering arrays and Taguchi techniques based test execution times operationalized as the variable of interest. Berling and Runeson (2003) researched test cases selection by applying fractional factorial covering techniques. Briand, Labiche, and He (2009) applied classical design of experiment techniques to test suite generation. Bryce and Colbourn (2006; 2007; 2009) studied software defect data generated from testing employing a pairwise test covering strategy. Chandramouli (2002) investigated testing improvements with test suites and test execution times using classical experimentation techniques. Forbes, Lawrence, Lei, Kacker, and Kuhn (2008) like Cohen et al. (2008) investigated IPO strategy for constructing covering arrays. Hoskins, Colburn, and Montgomery (2005) investigated improvements in software performance testing using covering arrays. Wallace and Kuhn (2001) explored software defects in test software-based medical devices. What was common for all of these research efforts was the fact that their studies showed improvements in the testing efforts that incorporated experimental designs. What was lacking in these same efforts was the fact that there was not conclusive evidence for

main stream software testing that the techniques utilized would work in any software testing domain.

Researchers (for examples, see Kuhn, Kacker, Lei, & Hunter, 2008; 2009) continue to look for greater improvements in software performance testing. In addition to just pairing factors for cause-effect relationships, the interactions between the factors were observed. It was shown that interactions among several factors by manipulating factor combinations using mathematical algorithms proved to be another way to assess software testing effectiveness. In this instance, testing effectiveness is measured by the number of defects discovered during the testing. This is the bases for combinatorial testing strategies and considered by many researchers as a step above or beyond pairwise testing strategies. As amazing as these results and techniques are, software testing professionals have not embraced them enough to give software users, or society in general, the peace of mind that one would think they engendered. Instead, these efforts continue to domain specific and seemingly isolated research efforts.

Researchers who conducted experiments specifically on the interactions of specific test factors included Kuhn, Wallace, and Gallo (2004), Kuhn, Kacker, Lei, and Hunter (2008; 2009), and Lei, Carver, and Kuhn (2007). These research efforts and the researchers are all associated with NIST. These NIST researchers conducted experiments and produced empirical data showing that combinatorial testing strategies are very effective at detecting defects involving the interaction of up to six test factors. An interesting observation from these researchers was the discovery that the smallest test suite possible might not always produce the most effective results if the included test

cases do not include higher strength interactions. This finding qualifying the resulting smaller test suite is in contrast to that of Bryce and Colbourn (2007) who equated smaller test suite sizes with improved performance testing without any particular specification for the composition of the resulting test suite. The implication here highlights the importance of selecting significant test factors as variables of interest for observing interactions leading to the detection of defects. This suggests that even with test covering arrays that test the greatest functionality of the software, if the right interactions between the right factors are missing then the testing could be less than effective. This also makes a good case for additional studies across multiple original studies could either validate the findings of these researchers or weed out some of these research results by invalidating the results.

Additionally, researchers such as Yilmaz, Cohen, and Porter (2006), Walker and Colbourn (2009) and Cohen, Dwyer, and Shi (2008) investigated software testing improvements by experimenting with test suites execution times and test execution costs based on combinatorial interaction test covering techniques. Bryce and Colbourn (2006) conducted interaction testing that differed from other interaction testing efforts in a couple of ways. First, they conducted interaction testing for pairwise test coverage. The interesting difference here is that many of the researchers conducting interaction testing, particularly Kuhn et al. (2008:2009) and other NIST researchers, interaction testing was conducted for some n -way combinatorial testing, where n is greater than two. A second distinction for the research of Bryce and Colbourn is that they adapted an interactive pairwise testing strategy where only one test was executed at a time. On the surface it

would not appear that much interaction could be observed executing only one test at a time. From their research, however, they found that pairing this strategy with other testing methods proved to be a very effective cost-benefit ratio for finding software defects.

Other specific software testing experimental researchers included Giannakopoulou, Bushnell, Schumann, Erzberger, and Heere (2011), who conducted formal software testing research. Here formal indicated testing based on sophisticated greedy algorithms. Similarly, Grindal, Offutt, and Andler (2005) formulated sophisticated mathematical models in their research efforts. Lazić and Velašević (2004) combined simulations with classical DOE strategies in their work. They found that combining simulation with test array covering was very effective at finding software defects early in the software test phase. Hartman and Raskin (2003) also developed mathematical algorithms in their testing investigations. In fact, they were among the earlier researchers to take this scientific approach to software testing. While the common theme for these researchers was improving software performance testing, what differentiated their work compared to the other research in the literature reviewed was the emphasis on the mathematical algorithms and mathematical rigor.

In recent year, as testing has become more technical in nature involving more and more mathematical algorithms and mathematical modeling, software testers are finding a good mathematical foundation is a good skill to possess. These continued efforts are an indication that testing problems still exist and that research is continuing to improve software testing effectiveness. The seeming niches for the research efforts is a

further indication of the gap that still exists in the literature for findings across multiple studies and domains that validate techniques that can be applied to improve software testing effectiveness.

Variables of Interest

The scope of the literature reviewed is further defined by the variables of interest for this study. Areas of interest impacting software performance testing were the focus of the original studies and among the criteria for inclusion in the meta-analysis. Test case design and testing execution times were both focus areas of the original studies reviewed. They were also the focus of this research study. It is much more cost effective to discover and fix software errors in an earlier stage rather than later stage of software testing. From a statistical perspective, the more testing performed the better the reliability delivered product. However, it could also be argued that more testing does not necessarily equate to a more reliable software product if the testing has not been performed adequately. Therefore, a better approach to testing is to identify techniques that detect more defects during the early testing stages. To accomplish this, careful attention should be given the selection of the variables of interest in the software testing process and designing test cases accordingly.

Industry characterizes effective software testing as that which maximizes the number of defects detected with the minimum time, cost, number of tests, and test execution time expended. The variables of interest for improving testing performance were driven by these quality expectations. For this quantitative study, the research variables of interests, as those in the original primary studies, included test execution

times, test suite sizes, test costs. A variety of methods for generating test suites for pairwise coverage arise from a number of different objectives to be addressed by the study. The research variables of interest and software performance test criteria from the review literature (Kacker, Kuhn & Lei, 2009; Parsa & Khalilian, 2010) included the following:

- The size of test suites.
- The amount of execution time to generate test suites.
- The consistency of test suites generated.
- The amount of testing time to execute the generated test suites.
- The accommodation of seeds and test constraints.

Reducing software testing time which directly impacts testing cost was a primary and common goal for the original studies included in this research effort. The smallest possible test suite that covered all possible n -way interactions which yielded the best test performance was often desired as each additional test case increases the total cost of testing (Bryce & Colbourn, 2007). For effective software performance testing, a reduced execution time to generate test suites is as important to the testing cost as the time spent actually executing the test suite. The question for this research is whether DOE techniques applied to software testing increase the effectiveness of the testing performance.

Summary

In summary, the primary focus of this chapter is the literature search and review. After defining the scope for this research study as outlined in Chapter 1, Chapter 2 can be

viewed as the real beginning of the research activity. This chapter defined the literature inclusion criteria and explored the strategies for actually searching various sources for peer-reviewed literature based on those criteria. The chapter further detailed how the literature review would proceed once potential candidates for inclusion in the study were retrieved. Details were presented characterizing the literature review essay as a composition comparing and contrasting the various researchers, the experimental testing strategies utilized, and the resulting quantitative findings from the original primary research.

Many best practices in the business world have their origin in university laboratories or government-funded research. As explored in this chapter, there have been research efforts that have reported empirical findings supporting improvements in software performance testing when experimental designs were utilized. Why then has this approach to software testing not caught on in the business community? What is missing in these efforts is solid validation of the findings. All of these research efforts are isolated efforts that span various application domains. This has proven insufficient to garner the general acceptance of applying these proven techniques in real world applications. In industry, quality improvements in software performance testing are improvements that minimize the time and cost derived from the total amount of tests to be executed together with the amount of time to conduct the testing while maximizing the number of defects found, all of which engender user confidence in the reliability of the software. The current research efforts have not led to this level of confidence. The gap in the literature is reflected by the lack of reports for concerted research efforts spanning multiple efforts,

in multiple domains validating the efficiency and effectiveness to be gained in software performance testing when experimental designs are part of the testing process.

This literature review provided the framework for the structure and design of this research to assess the impact of applying experimental design to software performance testing improvements. Key considerations in this literature review were the following:

- Literature inclusion and search strategies.
- Theoretical basis and meta-analysis.
- Relevance of historical studies to the recent studies and to this meta-analytical study.

The next chapter presents a detailed discussion on meta-analysis, the chosen research methodology.

Chapter 3: Methodology and Design

Introduction

The problem addressed in this research was the gap in the scholarly research literature of concerted, concentrated efforts quantitatively to validate measurable software testing performance improvements with objective statistical data across a group of selected studies. The literature reported studies where there were testing improvements. The findings, however, were from isolated individual studies. They provided insufficient scientific evidence of a general conclusion for the body of knowledge regarding research studies that have proven that statistically-significant gains in software performance testing result when DOE techniques are applied.

The purpose of this research was to assess the reported findings from the primary software performance testing studies against the findings from an aggregate of software performance testing studies and add to the current body of knowledge in the software testing community. This research has potential positive significance for the research community, the software testing profession, and society in general. For the research community, it is an addition to the body of knowledge, and for the software testing community, the technical aspect of the testing process adds an element of objectivity and respect to software testing profession. Finally, for society in general the potential for positive change is in the peace of mind for consumers and end-users regarding the quality of testing that their software products have undergone.

This chapter describes the research design, the data collection process, and data analysis methods used to assess the relationship between software performance testing

results and the application of experimental design techniques to the software testing process. The main focus of this chapter is the steps I followed in conducting this research within the framework of the meta-analytic process.

Meta-analysis was the research methodology in this research study. The framework in which meta-analyses are conducted is essentially the same as the research process. The procedural steps for both processes are so interconnected that the connection forms the basis for the organization of this chapter. This chapter is outlined according to the procedural steps for the research process, emphasizing the connection to the steps of meta-analysis process as appropriate. The discussion begins with defining the target population and explaining how the sampling process works, which for this research is equivalent to establishing inclusion criteria for the studies to be included in the meta-analysis.

Target Population

The target population is that populace to which a researcher seeks to generalize resulting investigative findings. Singleton and Straits (2010) postulated that the significant decisive factors in defining the target population for a research project were the research topic and the type of unit for analysis. This research utilized the meta-analytic procedure to combine and statistically analyze the results of an aggregate of original studies on software performance testing effectiveness. The unit of analysis for this research was each individual primary study included in the meta-analysis and also each subgroup treated as a unit. Hence the target population was defined as all studies covered in peer-reviewed literature that reported findings from investigations into the

effectiveness of software performance testing. The following paragraphs describe in detail the sample population used in this research. Discussed also are the sampling frame, the sample size, the original study inclusion criteria, and an exploration of the sample characteristics. Determining the eligibility criteria for including study findings for this research was an important step in the meta-analytic process as it is in the research process. This fact is just one of many reasons supporting the choice of the meta-analysis research method.

Sampling Design

Because of the number of studies and articles on improving software testing, it was not practical to conduct this research on the entire population of studies, so a smaller representation of the populace was selected for inclusion. Cleverly defining inclusion criteria not only made for more a reliable study based on valid and reliable data, but also factored in determining the sample size. For the research process, this activity equated to developing the sampling design. From the target population, the representative samples for this research were articles which described investigations in software performance testing where DOE techniques were applied. In the research process this phase is called the sampling design or sampling frame, where specific cases (original investigative studies, in this instance) were judged for sample selection based on the characteristics shared with the target population. The sampling frame was defined based on the operationalized definitions of the target population foundational to the original testing studies sample selected for inclusion. I followed the rules of meta-analysis selection criteria for the inclusion of primary studies in this research effort. Inherent in the meta-

analytic procedure, and a very important step in the research process, is identifying inclusion criteria. These criteria are important for guiding the selection of original studies to include in the research study to include in the meta-analytic procedure.

Sampling Eligibility Criteria

For this quantitative study using the meta-analysis research design, the sampling design was intrinsic in the very nature of this research method. In other words, the sampling design for meta-analysis entailed defining the selection study criteria that made an original study eligible for inclusion in this research study. This research was on the effectiveness of software performance testing where experimental design techniques are applied. For a study to be included in this research effort, two main criteria had to be met. The first criterion was that the candidate study, whether or not experimental design techniques were applied, must have investigated software performance testing effectiveness and reported findings of performance improvements. The second eligibility criterion was that the dependent variable must have been operationalized in terms of the number of detected software defects. Once these two criteria were met then additional selection criteria regarding the variables could be investigated; for example, the research method, time frame, and publication type.

Key variables for inclusion. The key variables of interest for this research included number of defects, phase in which defect was detected, defect detection rate, and testing hours. All of the primary studies for inclusion in this research involved the investigation of improvements in the effectiveness of software performance testing by applying DOE techniques. The improvements reported in the eligible study candidates

were operationalized in terms of the number of software defects found, phases in which the defects were detected, the rate of detection during testing, and the number of hours of actual test execution. The dependent variables of interest in the original studies are the same as the key dependent variables for this research.

Quantitative research design. Since this research was quantitative in design, it was important that each original study also employed a quantitative research method, statistical data analysis, and reported quantitative findings. For inclusion, the potential studies had to have investigated any statistical correlation between the test suites derived from applying DOE techniques utilizing pairwise and combinatorial testing coverage strategies and the resulting number of software defects identified during the testing.

Time frame of original studies. In an effort to include the latest research findings, recent studies published in peer-reviewed journals composed the prime considerations for inclusion in this research. However, earlier relevant published peer-reviewed literature was also reviewed for an understanding of the relationship of earlier research and findings to the more recent and current research efforts.

Publication type. To be a candidate for inclusion in this research, the findings had to be published in a peer-reviewed journal. The span of publication types ranged from the technical journals to technical reports to academic research publications.

Sample characteristics. With the target population defined and the sampling framework defined, the next step in this research study was determining the study sample size. From the review of the peer-reviewed literature in Chapter 2 for this quantitative study, I determined that the research community has devoted much effort to investigating

software performance testing techniques. The reliability and validity of this research study hinged heavily upon the sampling frame and the operationalized inclusion criteria to ensure the reliability of any statistical significance detected between the variables of interests across the included original studies.

Sample Size Calculation

If the results of a quantitative study are to be generalized to an entire population, then a sample size needs to be computed. Aczel and Sounderpandian (2006) recommended the following algorithm for calculating the minimum acceptable sample size for conducting this study. I calculated the sample size for the number of studies to include in this research using an estimated standard deviation of 0.1 or 10% variance in the variables of interest in the target population. According to Bartlett, Higgins, and Kotrlik (2001), the estimation of the variance in variables of interests is a critical step in sample size calculation, especially since it is just that, an estimate, and the researcher has no direct control. Additionally, in meta-analysis, the term *effect size* represents the strength or impact of a study. By computing the effect size for each study, it could be gauged if there were consistency in means across the included studies. The estimated standard deviation used in the minimum sample size calculation was influenced by these facts. Using the following formula, the minimum sample size for this research was 96 studies.

$$n = \frac{Z^2 pq}{D^2}, \quad (4)$$

where z is the interpolated value for the 95% confidence level

p is a best guess for the population proportion. In this study, for the population of software testing efforts, p , is an estimate for the proportion for the possible number of those efforts that applied DOE.

q is $1 - p$ (pq represents the population variance)

D is the allowable margin of error

Assumptions:

- Population size unknown but assumed > 30 .
- Population is composed of categorical data (findings from software testing where DOE techniques were applied and those where they were not applied) but proportions are unknown.
- 95% confidence level.
- α is 5%.
- $p = q = .5$
- Willing to accept a margin of error D of 10%

Using the minimum sample size algorithm above,

$$n = \frac{(1.96)^2(.5)(.5)}{(.10)^2}, \quad (5)$$

$$n = \frac{3.84(.25)}{.01}, \quad (6)$$

$$n = \frac{0.96}{.01}, \quad (7)$$

$$n = 96. \quad (8)$$

Therefore, the minimum number of original studies needed for this study was 96. An equal number of original studies where DOE techniques were applied to software

performance testing and of those original studies without the application of DOE techniques were included. To distinguish the two study types, the studies are listed in separate tables in Appendix C and Appendix D. Table C1 contains the list of studies without DOE techniques and the list of studies where DOE techniques were applied are listed in Table D1.

Research Design and Method

To restate the problem statement: There is a gap in the scholarly research literature of concerted, concentrated efforts quantitatively validating measurable software testing performance improvements with objective statistical data validating this assertion across a group of selected studies. The findings from isolated individual studies provide insufficient scientific evidence of a general conclusion in the body of knowledge regarding research studies that have proven that statistically significant gains in software performance testing result when DOE techniques are applied. To close this gap in the scholarly research, I used meta-analysis, statistically synthesizing results across multiple original software testing studies. The sequence of steps for conducting this meta-analysis, as suggested by DeCoster (2009, p. 4) is as follows.

1. Determine the theoretical correlation (i.e. define the problem or research question(s)) to study.
2. Gather original studies with findings relevant to the chosen correlation.
3. Select and code effect size statistics for the original studies to be synthesized.
4. Compute the effect size statistics and analyze the impact of the moderating variables of interest.

5. Understand and report the findings of the meta-analysis based on the data analyzed with attention to publication bias.

The following describes in detail how each step of the meta-analytic process was accomplished.

Variables of Interest Format Definitions

Examining the same variables of interest in the meta-analysis as those examined in the primary studies is one way to ensure the validity of the findings. For purposes of this research, the independent input or manipulated variables to which DOE techniques were applied are selected factors from the primary studies that were studied for their impact on software testing effectiveness. Software testing independent variables that were operationalized included:

- The experimental design techniques

The two categories of experimental design techniques (Classical and Taguchi), as defined by Antony (2006), were represented in this research and included the following;

- Classical
 - Factoring covering
 - Pairwise covering arrays
 - Combinatorial arrays
 - Orthogonal arrays
 - Other
- Taguchi approach

The dependent variables were the findings from the original included software testing studies. These findings, which operationalized the software performance testing improvements, were reported using various measurements. For this research, the dependent variables were:

- Defects detected
- Defect detection rate
- Phase defect detected
- Testing hours

The formats for the dependent variables of interest included the following:

- Number of defects
 - Variable: Ordinal
- Defect detection rate
 - Variable: Ratio
- Phase detected
 - Variable: Ordinal
 - Categories:
 - 1 – Prior to Coding
 - 2 – Unit level testing
 - 3 – Integration testing
 - 4 – Final acceptance testing
 - 5 – Regression testing
- Total test hours

- Variable: Ordinal

Data Collection Procedure

With the target population (published peer-reviewed articles that reported research findings for software testing effectiveness investigations) defined, the focus then moved to collecting the original studies. For meta-analysis, the best data collection approach, according to Singleton and Straits (2010), was to use multiple and complementary sources. In theory, though, this strategy could produce too many potential studies to be practical. However, this was not the case in this instance. For this research, recent peer-reviewed articles reporting findings from quantitative studies that investigated improvements in software performance testing effectiveness were the source of original studies for the meta-analysis. At this point in the research process, an application was submitted to the Institutional Review Board (IRB) to ensure ethical practices were observed in the collection and use of this data.

After the data (the original articles) were collected, the studies' characteristics were parsed in a format for ease of grouping, and transformed into a format for statistical calculations and analysis. Another way to think of this research step is in terms of assigning numbers to the variables of interest and study characteristic based on the research question(s) for input to the chosen research computational model(s). This step in meta-analysis is called coding. The next paragraph describes the coding process and presents a sample of the information that was encoded.

Coding

After the variables of interests were defined, the sample population of original studies for inclusion in this research study, and the strategy for collecting them developed, the characteristics and variables for the information from each of the original studies were encoded. The point of this step of the meta-analysis process was to format the data for each study to create a database in numeric format for computations and statistical analysis. The coded information included descriptive characteristics and effect sizes for each of the original studies.

Each study was coded using the unique author/year, citation-like notation. Using this scheme, each original study in the meta-analysis was cross-referenced and then easily and uniquely identified in the references. Since the original studies in the references included both those where DOE techniques were applied and those where DOE techniques were not applied, the studies were further distinguished accordingly in separate tables. See Table 3 for a sample of the characteristics and effect size that were coded for each of the original software testing studies included in the meta-analysis. Table 4 depicts a notional example of coded information for a combinatorial software testing study. The sample size is 10 software projects and all 10 projects were tested manually and tested using a software testing tool applying a pairwise testing technique. The encoded values or selected characteristics are in brackets.

Table 3

Sample of Software Testing Studies Characteristics That Can Be Coded

Sample Data to Code
Study Identification
Publication Type
Sample Size
Treatment group
Control group
Experimental design
Pairwise covering arrays
Combinatorial arrays
Orthogonal arrays
Taguchi approach
Effect Size
Effect Size Data Type
Mean and standard deviation
Treatment mean
Control mean
Treatment standard deviation
Control standard deviation
Significance Tests
t value or F value
p value

Table 4

Software Testing Application Coding Sample

Variable of Interest	Coded Value
Study Identification	Colburn, (2005)
Sample Size	10
Treatment group	5
Control group	5
Experimental design	
Factor Covering	
Pairwise Covering	X
Taguchi Approach	
Other	
Effect Size	.25
Effect Size Data Type	
Mean and standard deviation	
Treatment mean	.30
Control mean	.18
Treatment standard deviation	.01
Control standard deviation	.02
Significance Tests	
<p p="" value<=""></p>	.05
Number of Test Cases	50

Effect Size and Data Computations

With the included studies collected, first the reported findings were isolated. All of the included studies reported findings showing improvements in software performance testing. The CMA software package has a spreadsheet interface for data entry. The raw data of reported findings were entered for each study, as depicted in Appendix E and Appendix F. Point estimates were calculated for studies' findings data entered into the CMA software package.

There are many effect size statistics used in meta-analytic procedures, as attested by Lipsey and Wilson (2001). The effect size statistics supported by the CMA software include the following:

- Odds ratio
- Log odds ratio
- Peto odds ratio
- Log Peto odds ratio
- Risk ratio
- Difference in means
- Standardized difference in means
- Hedges' g
- Correlation
- Fisher's Z

However, in practice, only a few are often used. In this research, the Cohen's d (the standardized mean difference) effect size statistic, as shown in Table 4, was used. From

the original studies' data, the standardized mean difference was calculated for each study. The CMA software package computed the statistical calculations. A feature of the software package is customizing the effect size once an effect size statistic has been computed. For example, once the *standardized mean difference* effect size was calculated, the display could be customized to show the same effect size data as *correlations*. The formula used by the software package to convert from *d* to *r* format is shown in Appendix G. The formulas for computing *r* or *d* formatted data effect sizes manually are shown in the following paragraph.

The data computation for the standardized mean difference, *d*, was computed by dividing the standard deviation into mean difference for each study. The formula for computing a population's standardized mean difference is;

$$d = \frac{\mu_1 - \mu_2}{\sigma}, \quad (9)$$

where μ_1 and μ_2 are means and σ is the standard deviation. The data computation for the correlation (*r*), which is the relationship between variables, utilizes the formula

$$r_{xy} = \frac{\sum z_{xi}z_{yi}}{n}, \quad (10)$$

where *n* is the total original studies and *x* and *y* are variables with standardized measures z_{xi} and z_{yi} for case *i* (Borenstein et al., 2009). For each of the included studies, the input data and data computations for *d* are shown in Appendix H.

Variables and Hypotheses

The hypothesis testing was carried out during this step in the meta-analytic process. The research question: What is the relationship between the DOE techniques

applied to test case design during testing and the effectiveness of the software performance testing? The details of the hypotheses and variables of interest for this research are discussed in the following paragraphs.

Variables

The variables of interests are the dependent variables and independent variables from the original studies. These dependent variables operationalize the resulting findings, in the form of improvements or effectiveness measures, for the original research testing. This effectiveness is measured in terms of defects and how long it takes to complete the testing. Thus, the defects detected, the rate of defect detection, the phase of defect defection, and the total number of test execution hours are variables to capture the measure of effectiveness. The coding for these variables is presented in Table 6. The independent variables, the DOE techniques, operationalize factors that might possibly influence a relationship between test case design and the measure of effectiveness in the software performance testing. To establish a framework for these variables of interest, software performance testing and software effectiveness were defined in terms of the dependent variables.

Performance testing is concerned with the resulting focused testing performance and the test execution performance (Nirpal & Kale, 2011). For this type of software testing, soak testing for software endurance and stress testing are two examples of the focused testing. To capture performance testing productivity, Nirpal and Kale proposed the following algorithm.

$$\text{Software Performance Testing} = \frac{\text{Valid Defects during testing}}{\text{Total Detected Defects}} * 100\%. \quad (11)$$

Similarly, Whyte and Mulder (2011) suggested the following formula to capture software testing effectiveness:

$$\text{Software Testing Effectiveness} = \frac{\text{Number of defects found during testing}}{\text{Total number of Defects identified}} * 100\% . \text{ (12)}$$

Note that the variables of interest are the same for both software testing definitions.

While the definitions are similar, the authors focused on different aspects of software testing. Nirpal and Kale (2011) focused more on peak volume testing or break-it stress testing. Whyte and Mulder (2011), on the other hand, were more concerned with the same variables but tracking defects starting early and continued throughout the software development cycle. Both formulas relate to findings based on defects, central to operationalizing the dependent variables. The number of defects detected in the testing phase is a key element of software testing effectiveness, as shown by equation 12. The rate of defect detection is software testing effectiveness per some testing time frame or CPU time frame. The phase of defect deflection is software testing effectiveness broken down by the development or testing cycle. Additionally, Whyte and Mulder were interested in other traits that software testing encompasses, such as the following characteristics:

- The software testing is completed on schedule (all scheduled tests executed).
- The software testing results in a high number of detected defects.
- The software testing defect detection capacity in early phases of testing is high (i.e., high probability for early detection of the *hard-to-find* defects).
- The software testing has an increased percentage for finding defects per hour of testing

- The software testing process is cost effective.

While each variable of interest impacts test effectiveness, all of the following characteristics are covered in the hypotheses for this research. These characteristics are:

- The resulting number of defects detected during the software testing process.
- The defect detection rate as derived from the valid number of defects detected per hour of testing execution.
- The testing phase in which the defect is detected.
- The number of test cases in the test suite used in actual testing execution.
- The number of hours to complete testing execution.

The cost component of testing efficiency for both software performance testing and testing effectiveness can be useful metrics in the software testing industry. To evaluate test performance Nirpal and Kale (2011) submitted the following formula.

$$\text{Test Execution Performance} = \frac{\text{Number of Test Cases}}{\text{Test Cases Execution Hours}} * 8 \text{ hours}, \quad (13)$$

where test shifts are eight hours. The greatest cost saving is achieved when all scheduled test cases are executed in the shortest possible testing execution time. This formula provides the basis for the fourth dependent variable, *total testing hours*. A reduction in the number of test execution hours translates into increased testing effectiveness in terms of the test execution performance per equation 13.

The correlation between the variables of interest and test effectiveness was represented in terms of the effect size in the meta-analytic process. The data for these variables of interest came from the original studies included in this research. The standardized mean difference, Cohen's *d*, was calculated for each included study. (See

Table H1 and Table H2). The input data for the software package were Cohen's d and the subgroup sample size for each study. The meta-analysis was based on subgroup analyses where one subgroup was composed of those studies that applied DOE techniques and the second subgroup was composed of those studies that did not apply DOE techniques. The effect size for one subgroup was then compared to the effect in the second subgroup. The p value was the test statistic used for calculating the effects for each study and the z test was the test statistic for comparing the subgroups. The details for the steps of this portion of the research are discussed in Chapter 4.

The Hypotheses

I developed five hypotheses for this study to address the research question. The first hypothesis tested the effect sizes for the two subgroups to determine the relationship between the effectiveness of software performance testing and applying DOE techniques at the subgroup level. This hypothesis test encompassed all of the 96 original studies and any of the four dependent variables included in this research, 48 studies per subgroup. The dependent variables for this hypothesis were the findings or effectiveness measures. The subgroup of studies was treated as a unit. Hence, the total of all the reported findings were treated as the single effectiveness measure for the subgroup. The effectiveness measures were not differentiated per dependent variable. The remaining four hypotheses tested for the relationship based on a cluster of studies with a particular effectiveness measure of the reported findings, one hypothesis per effectiveness measure. Thus, the number of original studies in each of these hypotheses tests was based on the number of

studies that reported findings in the particular effectiveness measure (dependent variable).

Let P equal testing performance effectiveness without the application of DOE techniques, P_D equal testing performance effectiveness with the application of DOE techniques, and PT equal software test execution performance effectiveness defined by equation 13.

Overall effectiveness measure per subgroup. This hypothesis addressed the subgroup level.

H_{01} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing.

H_{a1} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing.

The corresponding mathematical notation for this hypothesis is as follows.

H_{01} : $P_D \leq P$ (application of DOE does not increase effectiveness)

H_{a1} : $P_D > P$ (application of DOE increases effectiveness)

Examining the research question in terms of the effectiveness measures, the hypotheses for this study were as follows.

Defects detected. This hypothesis addressed the dependent variable for the detected defects.

H_{02} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the total

valid number of defects detected during the software testing process to the total number of defects found.

H_{a2} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the total valid number of defects detected during the software testing process.

The corresponding mathematical notation for this hypothesis is as follows.

H_{02} : $P_D \leq P$ (application of DOE does not increase effectiveness)

H_{a2} : $P_D > P$ (application of DOE increases effectiveness)

Defect detection rate. This hypothesis addressed the dependent variable for the rate of the detection of the defects.

H_{03} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by number of defects detected per hour during the software testing process.

H_{a3} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured number of defects detected per hour during the software testing process.

The corresponding mathematical notation for this hypothesis is as follows.

H_{03} : $P_D / \text{hour} \leq P / \text{hour}$ (application of DOE does not increase effectiveness)

H_{a3} : $P_D / \text{hour} > P / \text{hour}$ (application of DOE increases effectiveness)

Phase detected. This hypothesis addressed the dependent variable for the software testing phase in which the defects were detected.

H_{04} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the number of defects detected during the earlier phases of the software testing process.

H_{04} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the number of defects detected during the earlier phases of the software testing process.

The corresponding mathematical notation for this hypothesis is as follows.

H_{04} : $(P_D)_n \leq P_n$ (application of DOE does not increase effectiveness)

H_{04} : $(P_D)_n > P_n$ (application of DOE increases effectiveness),

where $n = 1$ to n ($1 =$ development phase prior to coding, $2 =$ unit testing, $3 =$ integration testing, ..., and $n =$ system testing phase) denotes the testing phase for the reported study findings.

Testing hours. This hypothesis addressed the dependent variable for the total test execution hours.

H_{05} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the reduction in total number of hours to complete test execution during the software testing process.

H_{05} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the reduction in total number of hours to complete test execution during the software testing process

The corresponding mathematical notation for this hypothesis is as follows.

$H_{05}: PT_D \leq PT$ (application of DOE does not increase effectiveness)

$H_{05}: PT_D > PT$ (application of DOE increases effectiveness)

Testing the Hypotheses

For the first hypothesis, H_{01} , I ran a z test to test the overall relationship between the application of DOE techniques and the effectiveness of software performance testing at the subgroup level, based on a combination of all the studies and all the dependent variables. For each of the subsequent hypotheses, H_{02} through H_{05} , I conducted a t test to compare the effect size data of the subgroup that had DOE techniques applied versus the subgroup that did not apply DOE techniques on each of the effectiveness measures. See Appendix I for the data. Also note, from the statistical data presented in the figures of Appendix I, that the Z value, p value, and confidence interval are part of the analysis of the software package and were used in testing the hypothesis.

The Z value was used to test the null hypothesis assuming a true effect (mean) of zero. The statistical testing compared the effect of the two subgroups to determine if either is more effective than the other at improving software performance testing. The Z value in meta-analysis indicates the statistical significance of the effects between studies. This value was used in assessing the influence of the independent variables in improving software effectiveness.

A second statistic, Q , was part of the computational analysis of the software package test for heterogeneity. The analysis tested the subgroups as single units and computed the Q value. The Q value, a chi-square statistic, takes the number of studies and the degrees of freedom to assess the variance within studies and between studies. The

Q value together with the p value determined the variance between effect sizes for the subgroups to establish which was more effective in performance software testing.

Both the Z value and Q value shown in Appendix I resulted from two-tailed testing with a 95% confidence interval and the p value. Borenstein et al. (2009) suggested that there is a perfect relationship between the p value and the confidence interval. A p value > 0.05 indicated the lack of statistical significance against the null hypothesis. A failure to reject the null hypothesis then indicated that there was not sufficient evidence to support the claim that applying experimental design techniques to software performance testing improves software testing effectiveness. Conversely, a p value ≤ 0.05 was cause to reject the null hypothesis. The actual statistical calculations for this research were performed using the CMA software package. See Appendix A for a statement of the quality of this software package.

Data Analysis

Subgroup analysis was the meta-analytic approach used to statistically analyze the data for this research. One subgroup was composed of the original studies where DOE techniques were applied and the second subgroup was composed of the original studies that did not apply DOE techniques. Using the CMA software, the effect size (Cohen's d format; see Table 4) and effect size variance for each of the included studies was computed. The effect size statistic is strategic to determining the impact of the relationship of the variables of interest for each of the subgroups and ultimately this research. The software allows the use of a spreadsheet interface to enter the data (for example, number of defects detected, number of test cases, and number of hours of

testing) for effect size calculation. With the different types of software tested and different software domains for the included original studies, the computational features of the CMA software was fully utilized to run analyses to show each included study's impact on the combined effect from the set of multiple studies. The software package facilitated repeated runs and the ease for evaluating the impact of each study on the total effect of the group of studies. Adding one study at a time in the runs made possible a cumulative analysis. Conversely, starting with all of the included software studies and removing one study at a time facilitated the sensitivity analysis portion of this research for both the fixed effect and random effect meta-analysis models.

Fixed Effects Versus Random Effects

The CMA version 2 software package supported both the fixed effects model and the random effect model in meta-analysis. The models are based on different assumptions. The fixed effect model assumes the population effect size is the same in all studies. Any variability is attributed to the sampling design. The random effect models, on the other hand, considers heterogeneous factors and allows for variations in the effect sizes of the included studied (Borenstein et al. 2009). The results generated from the two models may differ. Based on this consideration, either the random effect model or the fixed effect model could have been the way to proceed. This research study utilized the fixed effect model to analyze the included studies.

The effect size calculations were based on the data formats of the original studies. There were two subgroups, with the difference being one had DOE techniques applied to

software testing studies and is the designated treatment group whereas the second group of software testing studies that did not apply DOE techniques.

Statistical Analysis

The meta-analysis data computations and computational analyses were performed by the CMA meta-analysis tool. See Appendix A for an idea for the validation conducted for the CMA meta-analysis software package. Appendix B provides testimonials from users in academia and industry who have used the tool.

Effects were computed by combining data across comparisons and treating these studies and the corresponding effect sizes as if independent. If studies yield data for two or more comparisons, the assumption of independence was unlikely correct. In such instances the standard error for the overall combined effect would likely be erroneously small, the confidence interval too narrow, and statistical significance tests likely to reject more often than the nominal significance level. For this research study, only one effect size per included study was assumed. Statistics were computed for the fixed effect model. The overall effect size was not assumed to be the same for both subgroups but computed by comparing the effect size data within each subgroup and between the subgroups.

Presentation of Results

The results of the meta-analytic procedure are one or more effect sizes which represent the average magnitude for the relationship studied. For this research study, the results may show the source of variation across the included original studies.

Furthermore, the reporting of the meta-analysis assumes that no other meta-analyses (i.e. random or mixed effects) have been performed on these included studies for the

effectiveness of software performance testing where experimental design techniques have been applied. Following this assumption, the observed effect sizes were reported using guidelines established by Cohen as discussed by DeCoster (2009) and presented in Table 5.

Table 5

Effect Size Magnitude Rule of Thumb

<u>Size of Effects</u>	<u>d (Cohen)</u>	<u>r (correlation)</u>
Small	.2	.1
Medium	.5	.3
Large	.8	.5

Note. From “Meta-analysis notes” by J. DeCoster , 2009, p.34. Adapted with permission.

Upon completion of the data computation and analysis, I used the CMA software to generate graphics, charts, and plots to present the results and aid in interpreting. The software is capable of producing statistics, funnel plots, scattered plots, and detailed reports. The resulting report made use of the plotting capability to generate funnel plots. The generated funnel plots utilized symbols for the original studies, appropriate weights, and the combined effect size.

The plots help to provide the statistics in context with weights included and anomalies highlighted. They allow the researcher to depict both effects sizes for the included primary studies and the summary effect for the meta-analysis. See Figure 1 for an example forest plot from the CMA software package using boxes to represent the effect size and relative weight. The confidence intervals track the precision of the effect sizes. The plot gives an immediate indication of the relative impact of each primary study on the meta-analysis by the width of the confidence interval and the boxes. Note that effect sizes, based on mean difference (g), are shown for the included studies and that the overall combined effect size for the meta-analysis is 0.419. The statistics are computed using the fixed effect model with a 95% confidence interval. Borenstein et al. (2009)

considered forest or funnel plots a must for the final report because they help ensure the validity of the statistics used in the meta-analysis and as well as help to prevent researcher bias. Features of the CMA software package were also used to assess publication bias.

Publication Bias

The findings of the meta-analysis are reported in a similar manner to those of the original studies. A potential problem with presenting results in the research community for those conducting meta-analyses is what Lipsey and Wilson (2001) called uneven reporting practices. This problem is manifested in resulting reports in the form of missing data or reports that are either too vague or too concise. The best way to address this bias, according to Borenstein et al. (2009) is to compare effect sizes in published studies to those in unpublished studies, if available. Since only published research was included, this option is not applicable for this research study. To gauge the impact that bias might have on this research, the CMA software package sensitivity analysis feature was utilized to explore various options or scenarios. This analysis strategy helped with knowledge of issues that might occur if different decisions are made or if additional data are available. Forest or funnel plots, as depicted in Figure1, are also useful for quantifying the potential for publication bias. For example, if the studies plotted formed a symmetrical funnel shape about the point 0.50, it would indicate the absence of any potential publication bias.

Meta Analysis

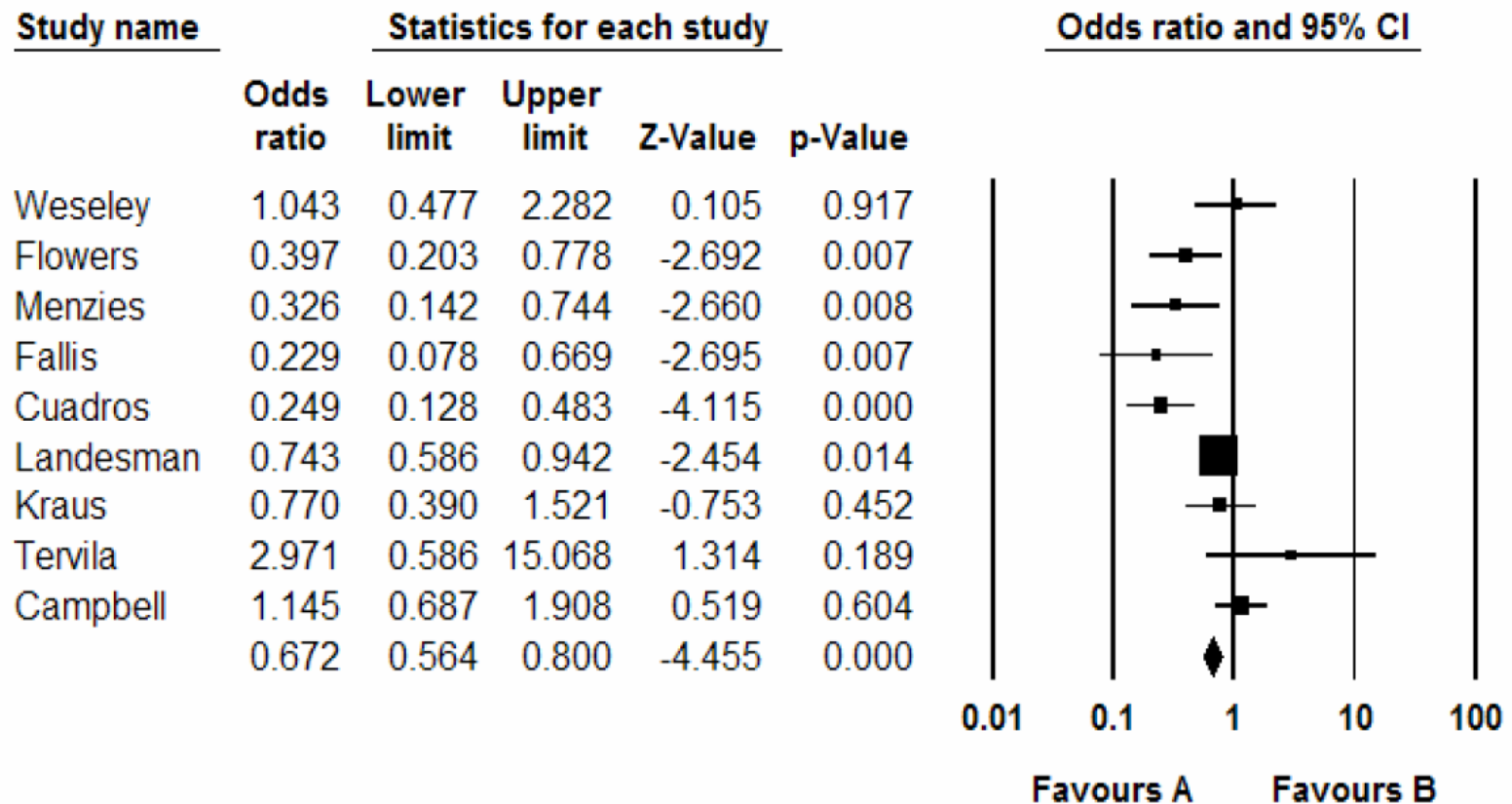


Figure 1. Meta-analysis results showing funnel plots and computed statistics for a fixed-effect model. From *Introduction to Comprehensive Meta-analysis*, by M. Borenstein, L. Hedges, J. Higgins, & H. Rothstein, 2009, West Sussex, UK: John Wiley and Sons, p. 7. Reprinted with permission.

Summary

In summary, this chapter presented a detailed discussion of the meta-analysis research method in terms of the steps to conduct the procedure. This research methodology was described from the perspective of the focus of this research, synthesizing reported findings from original studies that presented evidence of improvements in software performance testing when experimental design was applied. In so describing the meta-analytic process, the target population, sampling, the data collection process, the data analysis process, and how the findings would then be reported are also discussed. Note that these steps are all steps the researcher went through in the research process. These same research steps are inherent in the meta-analytic process, a process that addresses the limitations in other studies and statistically assesses effect or correlations observed among multiple studies. Moreover, for the gap under investigation, the meta-analysis research method was a perfect fit for assessing original findings for generalizability to all software testing efforts that utilize experimental design techniques.

The next chapter covers the actual performance of this dissertation project, to include the hypotheses testing, statistical computations and data analysis, and reporting of the findings.

Chapter 4: Research Results

This chapter contains a discussion of the findings from the software performance testing investigation conducted for this study. It is structured around the research question and the hypotheses that formed the bases for the study. The chapter begins with a brief review of the purpose of this research study, the research question, and the hypotheses. This review is followed by a discussion of the data collection procedure detailing how the included articles of the original studies were gathered, the data extracted, and that data organized and prepared for the meta-analysis. I discuss the meta-analysis conducted for this study, including the data analysis and presentation of the results. The results presented are organized by research question and hypotheses. The chapter concludes with a discussion addressing the research question and the hypotheses tested.

Introduction

The focus of this research was the efficiency and effectiveness of software performance testing. To briefly restate the purpose introduced in Chapter 1, the aim of this research was to evaluate the reported findings from the primary software performance testing studies against the findings from an aggregate of software performance testing studies and add to the current body of knowledge. The ultimate objective of this study was an assessment as to whether or not measurable improvements in the quality of software testing result from applying DOE techniques.

The key research question investigated in this study was as follows: What is the relationship between the DOE techniques (independent variable) applied to test case design and the effectiveness of the software performance testing (dependent variables)?

The first of the five hypotheses addressing this research question tested at the subgroup level by including all the studies in the two groups (48 studies with DOE applied and 48 studies without DOE techniques applied), as indicated in Table 1. For the 48 studies in the subgroup that did apply DOE techniques, not all of the four dependent variables were present in each individual study. However, for the DOE subgroup of 48 studies, all four of the dependent variables (defects detected, defect detection rate, defect detection phase, and testing hours) are included in this hypothesis.

H_{01} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing.

H_{a1} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing.

Recall from Chapter 3, equation 11, Nirpal and Kale (2011) defined software performance testing using the following formula;

$$\text{Software Performance Testing} = \frac{\text{Valid Defects during testing}}{\text{Total Detected Defects}} * 100\%. \quad (11)$$

Also, software testing effectiveness was defined by Whyte and Mulder (2011) (equation 12) as follows:

$$\text{Software Testing Effectiveness} = \frac{\text{Number of defects found during testing}}{\text{Total number of Defects identified}} 100\%. \quad (12)$$

The research question was examined from the perspective of the hypotheses tested. The dependent variables (defects detected, defect detection rate, defect detection phase, and testing hours) were operationalized in this study to answer the research question. I sought to assess whether there were any relationship between the independent variables (DOE techniques applied) and dependent variables measuring software testing performance improvements. In this study, the effect size data from the two subgroups (without DOE techniques and with DOE techniques) were synthesized and analyzed for a link between the DOE subgroup computed effect size and improvements in software performance testing, where the improvements were manifested as follows:

- Defects detected: Improved software quality as measured by more defects found in the overall testing process (i.e., sum total of all defects detected throughout all phases of the software development life cycle).
- Defect detection rate: An increase in test execution efficiency as assessed by the defect detection rate (for example, number of defects detected per hour).
- Defects detected by phase: Improved phase containment of defects, as measured by the number of defects detected in earlier phases in the software development life cycle. This translates into reduced cost, since it is cheaper to fix defects the earlier detected from both software correction and test time perspectives.
- Testing hours: A reduction in the total number of hours to execute all tests during the software testing process.

The remaining four hypotheses tested for this study corresponded, one-to-one, to the four dependent variables. These hypotheses presumed improvement in software performance testing efficiency and effectiveness when DOE techniques are applied with regard to the following four aspects of software performance testing:

1. H_{02} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the sum total of all the valid number of defects detected during the software testing process.

H_{a2} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the sum total of all the valid number of defects detected during the software testing process.

2. H_{03} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by number of defects detected per hour during the software testing process.

H_{a3} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured number of defects detected per hour during the software testing process.

3. H_{04} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the

number of defects detected during the earlier phases of the software testing process.

H_{a4} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the number of defects detected during the earlier phases of the software testing process.

4. H_{05} : The application of DOE techniques in the software test case design does not increase the effectiveness of the software performance testing, as measured by the reduction in total number of hours to complete test execution during the software testing process.

H_{a5} : The application of DOE techniques in the software test case design increases the effectiveness of the software performance testing, as measured by the reduction in total number of hours to complete test execution during the software testing process.

The CMA version 2 software package computed the results for the meta-analysis using the *analysis by subgroup* feature. The analysis synthesized findings across all 96 original studies, where 48 of the studies investigated the application of DOE techniques in software performance testing and the remaining 48 studies were software performance testing investigations that did not involve DOE techniques. With each of the 48 studies forming a subgroup, the software package treated each independent subgroup as the unit of analysis (study) in the meta-analysis. The software package computed an effect size for the subgroup of studies that had DOE techniques applied and an effect size for the

subgroup of studies where DOE techniques had not been applied. I performed a Z test on the subgroups effect size data to test the first hypothesis (H_{01}) to determine which subgroup had the more significant software performance testing improvement.

After computing the overall effect size for each subgroup and determining the statistical significance of the overall effect size for each subgroup, I used the software package to analyze the effect size data for the remaining four hypotheses. Each of these hypotheses was developed around one of the dependent variables. The dependent variables were operationalized as defects detected, defect detection rate, defect detection phase, or testing hours. The software package supports a function based on moderating variables. In meta-analysis, and this software package, moderating variables allow the grouping or categorizing of studies within a subgroup to see if the grouping influences the effect size of the subgroup. Assigning a variable label to a category of studies allows them to be entered into the software package as moderating variables. Moderating variables facilitate further subgroups comparisons by computing and comparing effect sizes based a defined category of studies in one subgroup to that same category of studies in a second subgroup. For this study, the moderating variables were categories defined by the dependent variables. The Q value was the key statistic used by the software package in the hypothesis testing, based on the moderating variables (dependent variables) in the hypotheses tested in this study.

Data Collection and Preparation

The primary objective of the data collection phase in this study was collecting articles of original software testing studies that were published in peer-reviewed journals. The first task in this data collection effort consisted of gathering original articles that assessed techniques for improving software performance testing effectiveness. I reviewed the articles with the research question and variables of interest in mind. Once the data for the variables of interest were obtained, the next step in the meta-analytic procedure was coding the variables for calculating the effect sizes. Coding is an important step in preparing the moderating variable of the hypotheses for the data analysis of the meta-analysis process. Before embarking on the actual meta-analysis, I analyzed the data to make sure they represented a thorough research of the literature articles for inclusion in an unbiased analysis. One of the key criteria I had for including a study was that it was peer-reviewed. A key criterion for meta-analyses, in general, was that they avoid publication bias. A couple of reasons for this criterion were (a) studies with statistically significant findings are more apt to be published and (b) most meta-analyses include published studies. Interestingly enough, these very reasons are cause for the concern that many in the research community have with the meta-analysis process. For some researchers, such as Borenstein et al. (2009), these reasons are thought to lead to publication bias. To address any potential publication bias, the analysis and preparation of the included data (original studies) for the meta-analysis, effect sizes were computed for each study and plots were generated. The plotted effect sizes for the studies provided

a visual means to determine whether the included studies had statistically significant findings. If the plotted studies appeared symmetrically dispersed about the mean then this was an indication that publication bias did not exist. This visual representation was a way to determine if some possibly needed studies were missing. This analysis was completed on all studies in both subgroups before proceeding with the data classification (see Figure 4).

Data Characterization

The collected articles were organized and categorized in an effort to ensure a thorough collection of articles covering all phases of the software testing process from industry and academia over a suitable period of time. The age range for the original articles collected for this dissertation is depicted in Figure 2. It gives the historical frame of reference for the interest in software performance testing. There were 96 studies, published between 1980 and 2013, included in this research. As shown, there has been a sharp growth in the number of published studies in the last ten years. Of these 96 included studies, 48 were studies involving the application of DOE in software performance testing. To continue the analysis, the original studies were categorized according to the software performance testing setting or the publication arena for the original studies. Figure 3 shows this breakdown of these included studies.

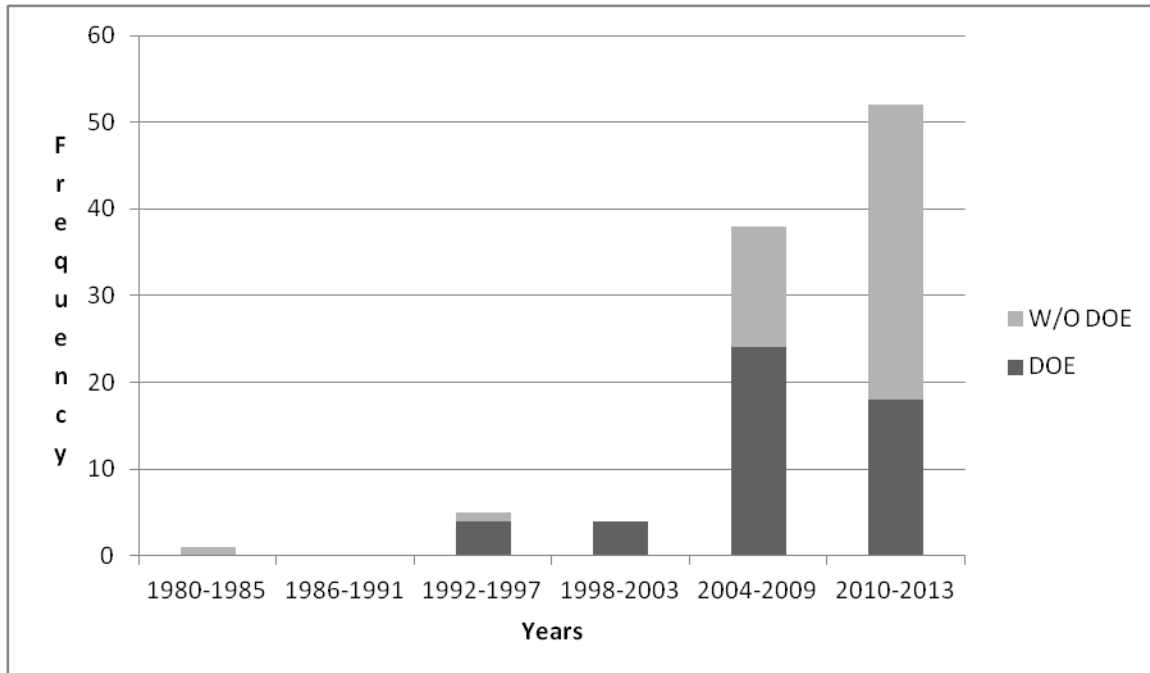


Figure 2. Year of publication for the included studies.

Publication for these studies peer-reviewed journals and technical reports were from academia, industry, research labs, and some were the product of collaboration efforts among the three. Studies that were first published in technical conferences or workshop proceedings were also included.

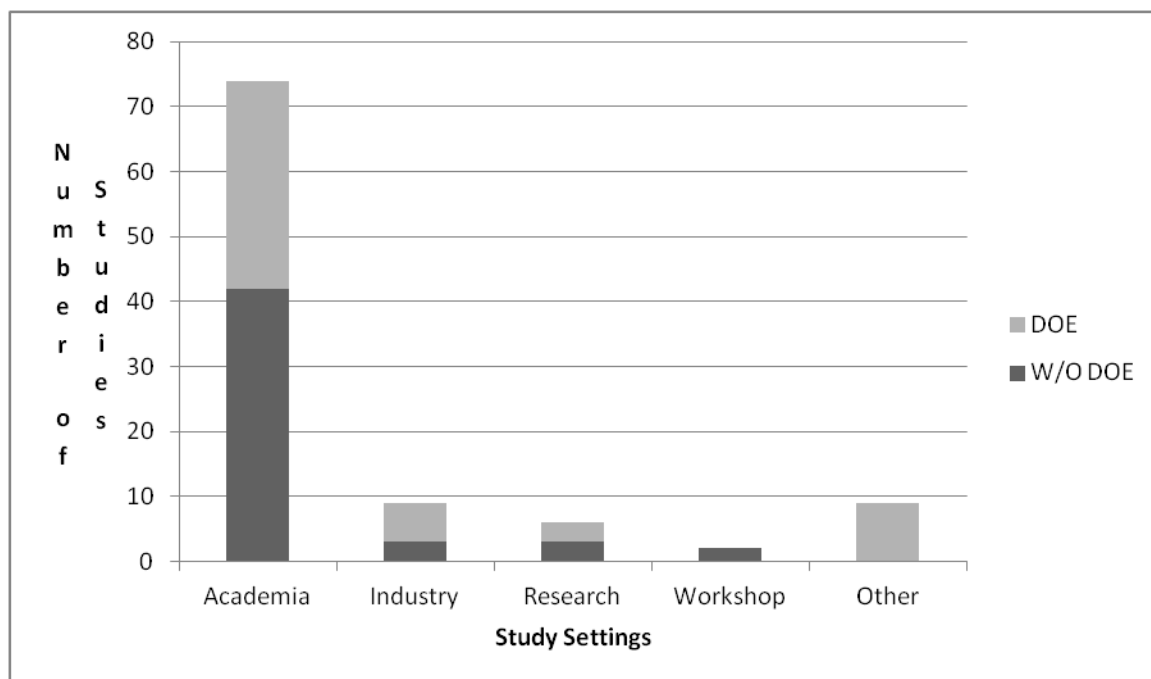


Figure 3. Publication profiles of original studies according to study setting. *Other* includes collaborative efforts among researchers from academia, industry, and research labs.

Publication Bias

Even with developing very concise inclusion criteria for the original studies, the availability of electronic databases, and the thoroughness of the researcher, the possibility for publication bias was a very real issue to be addressed. The reason for this concern stemmed from the fact that if important studies were not included, there would be the potential for a wider confidence interval and less powerful tests. Hence, any publication bias in the sample data for this research study would have been carried forward into the meta-analysis.

As mentioned above, the effect sizes from the included studies were plotted to get a sense of any tendency toward publication bias, from a visual perspective. Forest plots provide a graphical means for ascertaining any publication bias in the included studies. The plots display the data using either the log risk or relative risk. Appendix E depicts the forest plot for those included studies with DOE techniques applied and Appendix F shows those studies without DOE techniques applied. In both plots, the original studies are shown with the larger studies toward the top and the smaller studies toward the bottom. In addition to the forest plots shown in Appendix E and Appendix F, the figures also present the effect sizes calculated for each of the individual original studies. These point estimate effects sizes were calculated based on the results from each study and the sample size for each of the two subgroups, those that applied DOE techniques and those that did not apply DOE techniques. These forest plots present a graphical sense of the relationship between sample size and the effect size. Funnel plots are also designed to highlight the sample size, effect size relationship.

The funnel plots provided another visual means for assessing publication bias. According to Borenstein et al. (2009), the funnel plot highlights whether the effect sizes are consistent from study to study and indicates the precision (inverse of the standard error) for each study. The funnel plot in Figure 4 shows a vertical line at the summary effect. If the studies were shown clustered symmetrically about this line, it would be an indication that this study is publication bias free. Note, however, that the studies are

clustered slightly to the right of the vertical line. This asymmetry is an indication that there is a possibility of publication bias that needs to be addressed.

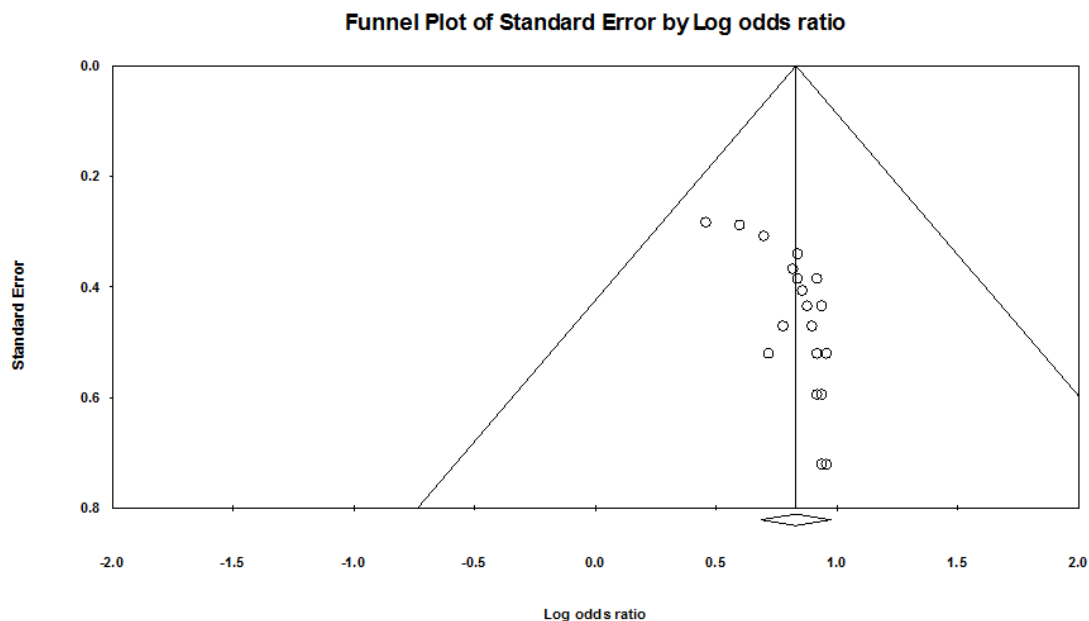


Figure 4. Funnel plot indicating the possibility of publication bias.

From the visual assessment of the collected studies, Borenstein et al. (2009) recommended that the researcher ask and answer the following questions.

- Is there evidence of publication bias?
- Is it possible that the combine effect is an indication of publication bias?
- How much of an impact would be imposed by any publication bias?

One issue posed by publication bias in meta-analysis is that there are possibly studies available but just missing from this analysis. Borenstein et al. (2009) described missing data imputation methods to correct this potential problem in meta-analyses. Data imputation is a process which allows missing data to be replaced by statistical values.

Trim and fill is a method that allows the missing data to be assigned. With this method, first it had to be determined where the missing studies would fall in the grouped studies. After determining where the missing studies should be in the group of already included studies, the studies need to be added and then the combined effect re-calculated. During the *trim and fill* analysis, the asymmetric studies are trimmed from the right side then these studies are filled into the missing slots by re-inserting both the trimmed studies and their counterparts. The results of a *trim and fill* analysis are depicted in Figure 5.

Trim and fill is just one of several methods for addressing missing data. Methods for imputing data range from assigning data based on an observed pattern from previously entered data values, to omitting the missing data, to very sophisticated statistical methods. The *trim and fill* analysis method discussed above is a feature implemented in the CMA software package to address publication bias. Figure 5 shows the same data as Figure 4 but it now includes the imputed data values.

Revisiting the questions that were suggested should be answered in assessing publication bias:

- Yes, there was a hint of the possibility for publication bias.
- Yes, it was possible that the combined effect would indicate publication bias.
- After reviewing the plot in Figure 4, together with the precision funnel plot in Figure 5, the inclusion of the imputed data appears to have minimal impact on the combined effect.

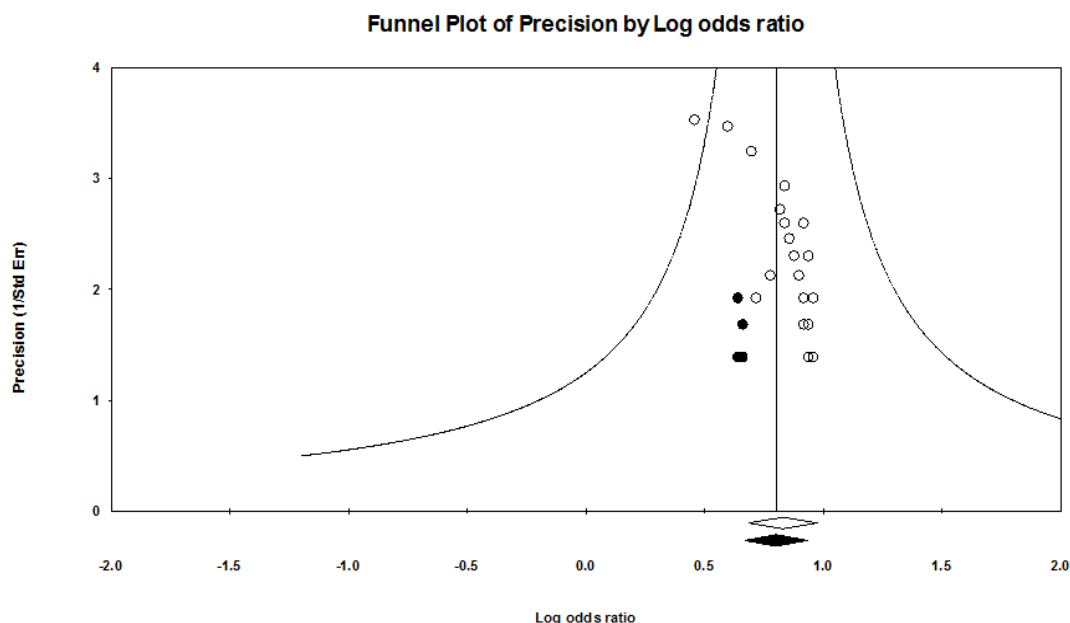


Figure 5. Precision funnel plot with collected and imputed data.

Once, the data for the variables of interest were collected, the data had to be prepared for the data computation and manipulation portion of the research. The findings from the original studies were examined for consistency in format and context for the variables of interest. The different study researchers used different measures for test effectiveness. This examination resulted in a coding scheme for the variables.

Data Coding Scheme

To continue the data preparation, the data had to be coded to ensure that all the variables of interest (dependent variables and independent variables) from all of the included original studies measured the same study characteristics and resulting findings. This coding is a very important step in the meta-analysis, since it centers on the collected data (variables of interest) from the original studies. Study variables of interest fall into

three categories, according to Lipsey and Wilson (2001): (a) independent variables from the original studies, (b) moderating variables, which may influence the findings, and (c) bibliographic data. Coding, however, distinguishes between the study characteristics and the resulting findings. In other words, the study characteristics are the independent variables and the study findings are the dependent variables. In this step, I made sure that all variables of interest across the group of original studies were measuring the same data. This addressed the research community's concern of synthesizing *apples-to-oranges*, which is another issue so often levied against the meta-analytic process. For this research the coding scheme devised is presented in Table 7. The hypotheses testing for the dependent variables was based on the variables of interests are shown in the Table 6. The Type shown is based on how the variables were used and reported in the original studies. In this study, the software testing performance improvements were operationalized in the variables of interest listed in Table 6. The independent variables are shown in Appendix E, Table E1.

Table 6

Variables of Interest

Variable	Type
Number of defects	ordinal
Defect detection rate	ratio
Phase detected	ordinal
Number of Test hours	ordinal

Associating these variables with the performance improvement measures reported in the original studies' findings led to the coding scheme used in the meta-analysis for this

research study. The study descriptors for the variables coded for this research are shown in Table 7. For this study, the variables of interest were dependent variables or findings reported from the original studies. All of the original studies operationalized test effectiveness in terms of either the number of defects detected or the resulting number of hours to complete the execution of the test cases. The perspective for the number of defects data reported in the original studies, however, varied from the explicit integer number of defects detected, to the percentage increase in the reported defects, to the defects reported during the specific stage of development or testing. Software efficiency, on the other hand, was operationalized in the original studies in terms of the dollars and hours spent per detected defect (Lazic & Velasevic, 2004). This efficiency measure was derived from the length of time it took to complete the testing as a function of the number of test cases executed or the reduction in test suites size that accounted for the actual test execution time. To restate from Chapter 3, equation 13 captured a formula for deriving testing efficiency mathematically based on test execution.

$$\text{Test Execution Performance} = \frac{\text{Number of Test Cases}}{\text{Test Cases Execution Hours}} * 8 \text{ hours} . \quad (13)$$

Because of the various ways software performance testing improvements were operationalized and measured across the original studies, scheme in Table 6 was devised to ensure that the original intent was preserved and all measures were represented.

Table 7

Coding Scheme

Features	Descriptors
DOE Techniques	Classical (Factor covering arrays, Pairwise covering arrays, Combinatorial arrays, Orthogonal arrays, etc.) Taguchi Approach
Effectiveness measures	- Defects detected (measured in terms of the increase in defects detected or sum total of all defects detected throughout all phases of the software development life cycle) - Defect detection rate (as assessed by the defect detection rate, for example, per hour or per software build) - Phase detected (measured by the identification of the phase in which defect is detected, emphasizing earlier phases in the software development life cycle) - Testing hours (measured by a reduction in the total number of test execution hours to test execute all tests during the testing process)
Testing duration	Short (< 8 hour) Intermediate (8 to 39 hours) Long (> 40 hours)
Testing study setting	Academia Lab Industry
Testers Proficiency	Low Intermediate High
Time Study Publication	Less than 5 years Greater than 5 years
Testing Type/Phase	Requirements / Unit / Integration / System
Publication Type	Peer-reviewed journal / Conference Workshop Proceeding

DOE Techniques

The DOE techniques covered in the original studies were generally of two categories, Classical or Taguchi, as described by Antony (2006). The Classical techniques included the factorial design or factor covering arrays approach which considers interactions between factors. For example, the factorial techniques were shown to be effective at addressing defects by increasing test coverage (Ahmed & Zamli, 2011; Bandurek, 2005; Berling & Runeson, 2003; Bryce & Colbourn, 2006). Examples of test case reductions, achieving at least or better defect detection, were shown in studies by Cangussu, Cooper, and Wong (2009), in an earlier investigation by Dalal and Mallow (1998), and a more recent investigation by Parsa and Khalilian (2010). Taguchi approaches were applied in studies that addressed efficiency and time reduction. An example is a study by He, Staple, Ross, and Court (1997). See Appendix E, Table E1, for the included studies data characteristics that show which studies employed which DOE technique(s).

Effectiveness Measures

As observed when developing the coding scheme, researchers in the original studies used different variables to measure the software testing improvement in terms of effectiveness or efficiency. Some researchers reported improvements in terms of the total number of defects, a percentage increase in the number of defects detected, or the rate at which defects are detected. Other researchers reported defect improvements based on the phase in which the defects were detected during the various software development phases

including those phases before coding began. Still others categorized the defects per the stages within the software testing process (for example, unit testing, integration testing, system testing or regression testing). So, whether the original software testing study focused on the total number of defects detected, the phase of defects detection, especially in early phases of the software development, the number of defects detected per software build testing, or reducing the number of hours for execution of test cases, the ultimate goal was increasing the software performance testing efficiency and effectiveness. See Table E1 and Table F1 for the raw data showing how the findings were measured and reported.

Testing Duration

A real concern for software testers in many testing situations is that of knowing when to stop test execution. Does the testing stop when the scheduled time allotted has expired? Does testing stop when all of the test cases have been executed? The issue of the testing execution time is directly correlated to the cost of testing. Several of the original studies addressed testing efficiency by targeting the testing execution time, especially for system testing that could range from hours to weeks (Devaraj, Kumar, Kavi Mallow, & Iannino, 2011; Forbes, Lawrence, Lei, Kacker & Khun, 2008; Ye, 2011). For studies where the testing was completed in less than 8 hours, the time duration was coded as a *short test time*. Any testing longer than 8 hours but less than 40 hours was deemed an *intermediate test time* while testing longer than 40 hours in duration was judged a *long test time*.

Software Testing Settings

As depicted in Figure 3, some of the original studies were conducted in academic environments, in industry, and in research labs. By far, most of the original studies were conducted by academics. The importance of this descriptor in the coding scheme is that it indicates the types of software testing studies that are taking place. In academia, most of the studies were based on mathematical models or were web-based testing. Examples include Alsmadi (2012) in academia, Watkins (1982) from industry, and Kuhn (2004) from the research lab.

Testers Proficiency

There was a significant correlation between the testers' proficiency and the study settings. Itkonwn, Mantyla, and Lassenius (2013) specifically addressed the improvement in software performance testing effectiveness that is realized due to the testers' knowledge of the software testing process, as well as any knowledge of the system under test, or knowledge based on a relationship with the customers and users. This tendency was evident in the testing that occurred in industry. On the other hand, there were studies that showed effectiveness improvements were also gained when the testers were not overly proficient in the testing process, had no real knowledge of the unit under test, or the end-users. This was really evident in academic settings where in many instances of unit testing, the testers were the software developers, such as in the study reported by Baharom and Shukue (2008).

Publication Timeline

Figure 2 presents the timeline for the included original studies. Of significance here is the fact that more than half of these studies were conducted in the last five years. This timeline can be seen as a reflection of society's increased dependence on technology and software base products. This dependence has caused an increased concern for the reliability and quality of software and software-based products. Also of interest is the fact that research into applying DOE techniques to software performance testing is not new, but has been occurring since the late 1990s, such as the research of Dalal and Mallows (1998).

Testing Phase

A fair representation of the original studies outlined findings where software testing performance improvements were realized when defects were detected early in the development process. In these instances, there was a resultant increase in the total number detected because the testers could vary their approach as they learned more about the system under test based on the number and types of defects detected. Such studies were categorized in the literature as adaptive testing studies. Original study examples included Hu, Jiang, and Cai (2009) and Kuhn et al. (2008).

Publication Type

The article by Ahmad, Khan, and Rafi (2010) was the only study that at the time of this study had only been published in a conference proceeding publication. The remaining 95 were all published in peer-reviewed literature, as was my intent.

Data Analysis

The analysis process covered several steps. As discussed in Chapter 3, using the CMA Version 2 software packages, the data analysis proceeded as follows.

- The first step was entering the study findings (raw data) as reported in the original studies. The data was entered as *event* and *sample size*. This step resulted in the funnel plots in Figure 4 and Figure 5 as well as the forest plots depicted in Figure E1 for the studies that had DOE techniques applied and Figure E2 for studies that did not apply DOE techniques.
- A cumulative analysis was run on the data to check for publication bias. Funnel plots of the data are shown in Figure 4 and Figure 5.
- The applicable effect size statistics to use in a meta-analysis depend on the nature of the study findings being synthesized as well as the research question(s) and hypothes(es) being tested. In this study, the research question and hypotheses were about the relationship between the improvement in software performance testing and applying DOE techniques. The original studies, however, did not report correlation data. Using correlation effect size statistics, r , for such studies is preferable to standardized mean difference, d , effect size statistics, according to Lipsey and Wilson (2001), for a couple reasons. For one thing, d , has a tendency to weaken the strength of the observed relationship. For another, r , is a standardized index for a meta-analysis statistic and it is easy to convert between the two effect size statistics.

So I made the decision to work with *correlation effect sizes* as this measure better lent itself to the research question under study. I started by computing d , using the data in the original studies. Using a feature of the software package, the computational analysis computed and reported the effect sizes in r .

- In this research neither d nor r was reported in the included studies, but I was able to calculate d from the information in the studies and then use the software package to convert d to r . The first step of the computational analysis of the meta-analysis began with the calculation of the Cohen's d (standardized mean difference) effect size for each original study. A derivative of equation 9, ($d = (\mu_1 - \mu_2)/\sigma$) was used to calculate d . Since the subgroups are independent groups, the standard deviation had to be a within-group standard deviation calculated across subgroups. The formula used to calculate the within-group standard deviation, S_{within} , is

$$S_{within} = \sqrt{\frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}}, \quad (14)$$

where n_1 and n_2 are sample sizes from two independent data sets and S_1 and S_2 are the corresponding standard deviations. A derivative of equation 9, ($d = (\mu_1 - \mu_2)/\sigma$) was used to calculate d . Since the subgroups are independent groups, the standard deviation had to be a within-group standard deviation calculated across subgroups.

$$d = \frac{\bar{X}_1 - \bar{X}_2}{S_{within}}, \quad (15)$$

where \bar{X}_1 and \bar{X}_2 are the sample means. See Appendix H, Tables H1 and H2 for the raw data. The data are taken from the original included studies.

- Next, the computed effect sizes for the original were entered into the CMA software package for subgroup analysis. The data were entered for two group comparison; post data only meaning the two groups did not contain pre-test data and post-test data but were independent. Also entered were data for the sample size for each subgroup and the flag set to automatically determine the direction of the effect size (i.e., positive, negative, etc.).
- I performed the meta-analysis in the CMA software package based on the entered data with the following computational options:
 - Grouped by subgroup (no DOE techniques applied and DOE techniques applied) and set to perform an analysis across the studies within subgroup.
 - Set the computational option to generate correlation, r , (effect size statistic) for comparing the two subgroups. (The software package can convert from one effect size measure to another. In this study, from Cohen's d to correlation, r)
 - Used the effectiveness measures (total defects, defect detection rate, defects by phase, and total testing hours) as moderating categories.
- The computational analysis returned the correlation effect size, confidence interval, Q value, Z value, and p value for each study, per effectiveness measure, per subgroup. The software package generated the correlation effect

sizes for each study within each subgroup based on the original data, then synthesized the effect sizes for each study across each subgroup. The resulting synthesized effect size for the subgroup without DOE technique applied was then correlated with the synthesized effect size for the subgroup with DOE technique applied to come up with the overall correlation effect size between the subgroups.

- These correlation effect sizes were also plotted to show graphically which subgroup had the more statistically significant effect size in the analysis.

Assessing The Hypotheses

The Z test was performed on the two subgroups of all the included studies to test the statistical significance of applying DOE techniques at the subgroup level. The results showed that the subgroup comprised of studies that did have DOE techniques applied in software performance testing were more effective in software performance testing than the subgroup of studies that did not have DOE techniques applied. Table 8 shows the Z distribution test statistics from all 96 studies grouped by whether or not DOE techniques were applied (48 studies with DOE and 48 studies without DOE) to the software testing studies. Table 1 depicts all five hypotheses by dependent variable and the number of studies in each subgroup. For the first of the five hypotheses, the outcome of the Z test resulted in the rejection of the null hypothesis.

Table 8

Fixed effect model: Overall Results

Model	SubGroup	Effect Size	Standard Error	Lower Limit	Upper Limit	Z value	p value
Fixed	DOE	0.540	0.032	0.521	0.559	44.27	0.000
	NoDOE	-0.064	0.032	-0.091	-0.037	-4.66	0.000

Testing the first hypothesis at the subgroup level, inserting the effect size results from the Z test,

$$H_{01}: P_D \leq P.$$

$$H_{a1}: P_D > P.$$

Based on this test, I rejected the null hypothesis that the application of DOE techniques does not increase testing effectiveness. Additionally, as shown in Table 8, $Z = 44.27$ for the DOE subgroup and -4.66 for the NoDOE subgroup indicates that applying DOE techniques has more impact on improving software performance testing than not applying DOE techniques. Moreover, the 95% confidence intervals for the two subgroups do not overlap. For the DOE subgroup, the computed confidence interval is (0.521, 0.559). For the subgroup of studies without DOE techniques, the computed confidence interval is (-0.091, -0.037). From these statistics, with a p value < 0.0001 , the null hypothesis, H_{01} , (which stated that the application of DOE does not increase effectiveness in software performance testing) is rejected.

For each of the hypothesis tests based on a single effectiveness measure for this study, the Z test, the Q test, and the t test were some of the test statistics generated by the

software package. The CMA tool analyzed the data using the effectiveness measures of defects detected, defect detection rate, phase in which the defect was detected, and total hours of test execution. The analysis treated each effectiveness measure as a moderating variable, testing its influence on the final result.

Defects Detected

For the number of detected defects, 30 of the original studies reported this moderating measure of effectiveness for the software performance testing improvement. See Appendix I, Figure I1 for the computed statistics generated from the meta-analysis in the software package. The computed summary statistics from the meta-analysis computational analysis ($P_D = 0.142$, Z value = 4.717, confidence interval (0.084 to 0.200), and $p < 0.001$ for both Z value and Q value) for the DOE subgroup and ($P = -0.170$, Z value = -0.183, confidence interval (-0.210, -0.130), and $p < 0.001$ for both Z value and Q value) for the subgroup that did not apply DOE are reported in Table 9. The corresponding forest plot for these computed statistics is shown in Appendix I, Figure I2. The subgroup of studies where DOE techniques were applied is shown as group A and the subgroup of studies where DOE techniques were not applied is shown as group B. The meta-analysis computational analysis results, in the CMA software package, showed that the impact of the computed correlation effect size for subgroup A was more statistically significant than the computed correlation effect size for subgroup B.

Subgroup A is the subgroup of studies where DOE techniques were applied. Note that the overall result is shown on the very last line of the meta-analysis summary

statistics and denoted by the diamond in Appendix I, Figure I2. The strength of the testing performance effect size for the studies without the application of DOE, P , was weaker than the computed effect for the studies that did apply DOE techniques, P_D .

Testing the null hypothesis for the detected defects, inserting $P_D = 0.142$ and $P = 0.170$, the test resulted in $P_D > P$. Given

$H_{02}: P_D \leq P$ (application of DOE does not increase effectiveness)

$H_{a2}: P_D > P$ (application of DOE increases effectiveness),

the null hypothesis is rejected. The hypothesis testing showed that the application of DOE techniques in the software test case design did increase the defects detected during software performance testing process.

Table 9

Moderator Analysis: Defects Detected Summary Statistics

SubGroup	Number Studies	Effect Size (Correlation)	95% CI		Z	p	Q
			Lower Limit	Upper Limit			
DOE	10	0.142	0.084	0.200	4.717	0.000	391.678
NoDOE	20	-0.170	-0.210	-0.130	-0.183	0.000	1456.410
Overall	30	-0.070	-0.104	-0.036	-4.050	0.000	1920.887

Taking the computed correlation effect sizes from Appendix I, Figure I1 (Correlation column) and using as input data, the t test was conducted on the defects detected data statistics computed from the meta-analysis. Table 10 shows the resulting t test statistics. The sample size difference for the DOE subgroup in Table 9 and Table 10 is the result of an outlier in the DOE data, as shown in Figure I1, which was omitted for

the t test. The outlier is a negative number, -0.836, while all of the other data points are positive.

Table 10

Statistics for t test on Defects Detected Data

Assumed equal variances			Assumed unequal variances		
	NoDOE	DOE		NoDOE	DOE
Sample Size (n)	20	9	Sample Size (n)	20	9
Mean (\bar{x})	-0.140	0.293	Mean (\bar{x})	-0.140	0.293
Std. Deviation (s)	0.484	0.230	Std. Deviation (s)	0.484	0.230
Test Statistic	-2.535		Test Statistic	-3.259	
df	27		df	26	
p value	0.0174		p value	0.0016	
95% CI	(-0.7824 , -0.0824)		95% CI	(-0.7051 , -0.1597)	

Levene's test verified if the variances are equal. Given,

H_0 : Variances are equal

H_a : Variances are unequal,

the test resulted in the p value = 0.037. Since the p value < 0.05, equal variances are not likely for the included studies. The null hypothesis of equal variances is rejected, so unequal variances are assumed for the defects detected data.

Assuming unequal variances, the computed t test statistics for the confidence interval (-0.7051, -0.1597) where test statistic $t = -3.259$, $df = 26$, and the p value = 0.0016. From these results ($p < 0.05$) and a confidence interval that does not include zero, the null hypothesis, which stated that the application of DOE techniques in the software test case design did not increase the number of defects detected during software

performance testing process, is rejected. See Appendix I for the software package generated statistical data in Figure I1 for the defects detected data.

Defects Detection Rate

The computed statistics generated from the meta-analysis in the software package are shown in Appendix I, Figure I3. The summary statistics computed in the meta-analysis for the original studies that reported software testing performance improvements when DOE techniques were applied grouped by the defect detection rate are shown in Table 11. The statistics computed for this moderating effectiveness measure in the DOE subgroup ($P_D = 0.235$, Z value = 6.139, confidence interval (0.161, 0.306), Q value = 213.975, and $p < 0.001$ for both Z value and Q value) and for the studies without DOE subgroup ($P = 0.361$, Z value = 7.312, confidence interval (0.270, 0.446), Q value = 331.214, and $p < 0.001$ for both Z value and Q value). The corresponding software-generated forest plot for these computed statistics is shown in Appendix I, Figure I4. The subgroup of studies where DOE techniques were applied is shown as group A and the subgroup of studies where DOE techniques were not applied is shown as group B. The meta-analysis computational analysis results, from the CMA software package, showed that the absolute value of the computed correlation effect size for subgroup B was more statistically significant than the computed correlation effect size for subgroup A. The overall result is shown on the very last line and denoted by the diamond. The result of the computational analysis is a summary effect size for group A and a summary effect size for group B. The two effect sizes are compared and synthesized resulting in the combined

or overall effect size. In Figure I4, note that the overall effect size is depicted on the right side of zero indicating group B, the subgroup of studies that did not have DOE techniques applied, had the greater statistically significant impact on the effectiveness of software performance testing.

Table 11

Moderator Analysis: Defect Detection Rate Summary Statistics

SubGroup	Number Studies	Effect Size (Correlation)	95% CI		Z	p	Q
			Lower Limit	Upper Limit			
DOE	6	0.235	0.161	0.306	6.139	0.000	213.975
NoDOE	3	0.361	0.270	0.446	7.312	0.000	331.214
Combined	9	0.282	0.225	0.337	9.303	0.000	549.791

Testing the null hypothesis for defect detection rate for $P_D/hr = 0.235$ and $P/hr = 0.361$, the test resulted in $P_D/hr \leq P/hr$ for,

H_{03} : $P_D / hr \leq P / hr$ (application of DOE does not increase effectiveness)

H_{a3} : $P_D / hr > P / hr$ (application of DOE increases effectiveness)

From the resulting effect size statistics, the null hypothesis, which stated that the application of DOE techniques in the software test case design did not increase the number of defects detected during software performance testing, could not be rejected.

Table 12

Statistics for t test on Defect Detection Rate Data

	Assumed equal variances		Assumed unequal variances		
	NoDOE	DOE		NoDOE	DOE
Sample Size (n)	3	6	Sample Size (n)	3	6
Mean (\bar{x})	0.091	0.129	Mean (\bar{x})	0.091	0.129
Std. Deviation (s)	0.758	0.495	Std. Deviation (s)	0.758	0.495

Test Statistic	-0.092	Test Statistic	-0.078
df	7	df	2
<i>p</i> value	0.929	<i>p</i> value	0.945
95% CI	(-1.012, 0.936)	95% CI	(-2.112, 2.037)

The *t* test was computed for the defect detection rate data. Table 12 shows data where equal variances are assumed and data where unequal variances are assumed.

Levene's test verified if the variances are equal. Given hypotheses,

H_0 : Variances are equal

H_a : Variances are unequal,

the test resulted in the *p* value = 0.38. Since the *p* value > 0.05, equal variances are likely for the included studies. The null hypothesis of equal variances is not rejected, so equal variances are assumed for the defect detection rate data. The computed test statistics, within a 95% confidence interval of (-1.012, 0.936), were $t = -0.092$, $df = 7$, and the *p* value = 0.929 are shown in Table 12. From these results (larger *p* value, i.e. > 0.05), the null hypothesis, indicating that the application of DOE techniques in the software test case design did not increase the rate of detecting defect during the software performance testing process, could not be rejected. See Appendix I, Figure I3, for the defect detection rate data.

Phase Detected

The computed statistics generated from the meta-analysis in the software package are shown in Appendix I, Figure I5. The summary statistics computed in the meta-analysis for the original studies that reported software testing performance improvements when DOE techniques were applied grouped by the defects detected by phase effectiveness measure are shown in Table 13. The statistics computed for this moderating

effectiveness measure in the DOE subgroup ($P_D = 0.089$, Z value = 3.067, confidence interval = 0.032 to 0.145, Q value = 57.533, and $p < 0.002$ for the Z value and approaching zero for the Q value) and for the studies without DOE subgroup ($P = 0.542$, Z value = 16.146, confidence interval (0.488, 0.592), Q value = 1204.324, and $p < 0.001$ for both Z value and Q value). The meta-analysis in the software package demonstrated the subgroup that did not have DOE techniques applied to be more effective, statistically, in improving software performance testing.

Table 13

Moderator Analysis: Defects Detected By Phase Summary Statistics

SubGroup	Number Studies	Effect Size (Correlation)	95% CI		Z	p	Q
			Lower Limit	Upper Limit			
DOE	12	0.727	0.700	0.752	32.976	0.000	157.533
NoDOE	6	0.542	0.488	0.592	16.146	0.000	1204.324
Overall	18	0.670	0.645	0.694	36.096	0.000	549.791

The forest plot for these defects detected by phase computed statistics from the meta-analysis is shown in Appendix I, Figure I6. The subgroup of studies where DOE techniques were applied is shown as group A and the subgroup of studies where DOE techniques were not applied is shown as group B. The software package analysis revealed B, the subgroup of studies where DOE techniques were not applied, had more impact on the effectiveness of software performance testing. The overall correlation effect size result is shown on the very last line and denoted by the diamond is 0.276

Testing the null hypothesis for defects detected by phase using $(P_D)_n = 0.727$ and $P_n = 0.542$,

$H_{04}: (P_D)_n \leq P_n$ (application of DOE does not increase effectiveness)

$H_{a4}: (P_D)_n > P_n$ (application of DOE increases effectiveness),

resulted in $(P_D)_n > P_n$. Since $0.727 > 0.542$, the null hypothesis is rejected applying the fourth hypothesis, DOE is more effective. The confidence intervals, (0.700, 0.752) for the DOE subgroup and (0.488, 0.592) for the subgroup where DOE techniques were not applied do not overlap. There is a statistical significance indicating that the DOE subgroup is better than the No DOE subgroup at improving software performance testing.

The t test was computed for the defects detected by phase data from Figure I5 in Appendix I. (See Table 14 for the input statistics). Levene's test verified if the variances are equal. Given,

H_0 : Variances are equal

H_a : Variances are unequal,

the test resulted in the p value = 0.14. Since the p value > 0.05 , equal variances are likely for the included studies. The null hypothesis of equal variances is not rejected, so equal variances are assumed for the defects detected by phase data.

Table 14

Statistics for t test on Defects Detected By Phase Data

Assumed equal variances			Assumed unequal variances		
	NoDOE	DOE		NoDOE	DOE
Sample Size (n)	6	12	Sample Size (n)	6	12
Mean (\bar{x})	0.057	0.153	Mean (\bar{x})	0.057	0.153
Std. Deviation (s)	0.572	0.340	Std. Deviation (s)	0.572	0.340
Test Statistic	-0.449		Test Statistic	-0.377	
df	16		df	6	
p value	0.660		p value	0.719	
95% CI	(-0.548, 0.356)		95% CI	(-0.716, 0.525)	

From the t test conducted within a 95% confidence interval of (-0.548, 0.356), the computed test statistic $t = -0.449$, $df = 16$, and the p value = 0.660. With $p > 0.05$ and overlapping confidence intervals, the null hypothesis cannot be rejected, indicating that the application of DOE techniques in the software test case design did not increase the number of defects detected by phase during the software performance testing process. However, applying the fourth hypothesis, $(P_D)_n > P_n$, which indicates that the null hypothesis can be rejected. Conflicting indicators warranted further investigation. Utilizing the t test table for critical values of the t distribution, the critical t value for $df = 16$ and a 95% confidence interval is 1.746. Since the computed test statistic $t = -0.449$ is less than the t test table value, the results indicate that there is no statistical difference between the means of DOE and the NoDOE subgroups. See Appendix I, Figure I5, for the statistical data for the defects detected by phase.

Testing hours

From the testing hours resulting meta-analysis statistics in Appendix I, Figure I7 and the summary statistics from Table 15, $PT_D = 0.632$, Z value = 36.146, confidence interval (0.607, 0.656), Q value = 57.533, and $p < 0.001$ for both Z value and the Q value and for the studies without DOE subgroup, $PT = -0.258$, Z value = 1141.988, confidence interval = -0.298 to -0.216, Q value = 352.92, and $p < 0.001$ for both Z value and Q value). The forest plot (See Figure I8) shows a graphical representation of the meta-analysis results. The computed correlation effect size results show that the effect size for

subgroup B was more statistically significant than the effect size for subgroup A. The computational results of the meta-analysis illustrated that the effect size of the subgroup of studies where DOE techniques were applied was more statistically significant than the effect size of the subgroup that did not have DOE techniques applied.

Applying the computed correlation effect sizes, $PT_D = 0.632$ and $PT = -0.258$ to test the null hypothesis for this study, for

$H_{05}: PT_D \leq PT$ (application of DOE does not increase effectiveness)

$H_{a5}: PT_D > PT$ (application of DOE increases effectiveness),

resulted in $0.632 > -0.258$. Hence, the null hypothesis was rejected for this test and the alternative hypothesis, which stated that the application of DOE techniques did increase software performance testing effectiveness when the improvement was reported in terms of the total hours for testing execution, resulted in a more statistically significant effect size.

Table 15

Moderator Analysis: Testing Hours Summary Statistics

SubGroup	Number Studies	Effect Size (Correlation)	95% CI		Z	p	Q
			Lower Limit	Upper Limit			
DOE	20	0.632	0.607	0.656	36.146	0.000	1141.988
NoDOE	19	-0.258	-0.298	-0.216	-11.719	0.000	352.920
Combined	39	0.277	0.249	0.304	18.725	0.000	2588.119

Table 15 depicts the results for the t test computed from the testing hours data shown in Figure I7. Verifying the equality of variances using Levene's test,

H_0 : Variances are equal

H_a : Variances are unequal,

the test resulted in the p value < 0.0001 . Since the p value < 0.05 , equal variances are not likely for the included studies. The null hypothesis of equal variances is rejected, so unequal variances are assumed for the testing hours data.

Table 16

Statistics for t test on Testing Hours Data

Assumed equal variances			Assumed unequal variances		
	NoDOE	DOE		NoDOE	DOE
Sample Size (n)	19	20	Sample Size (n)	19	20
Mean (\bar{x})	-0.391	0.408	Mean (\bar{x})	-0.391	0.408
Std. Deviation (s)	1.040	0.360	Std. Deviation (s)	1.040	0.360
Test Statistic	-3.238		Test Statistic	-3.172	
df	37		df	22	
p value	0.0013		p value	0.0022	
95% CI	(-1.299 , -0.299)		95% CI	(-1.321, -0.276)	

From the assumed unequal variance results in Table 15, note that the computed test statistic $t = -3.172$, $df = 22$, and the p value = 0.0022 with a confidence interval of (-1.321, -0.277). The t test resulted in $P < .05$ and a confidence interval that did not include zero, which indicated that the null hypothesis should be rejected. Hence, showing the application of DOE techniques in the software test case design did reduce the total number of hours to complete test execution during the software performance testing process. See Appendix I for the statistical data by effectiveness measure and Figure I7 for the data for the testing execution hours.

Key Findings

The meta-analysis statistics shown in Table 8 summarize the key findings by subgroup. The forest plots generated in the meta-analysis for this study are crucial in

understanding the research findings. The forest plots provide a good pictorial representation that aids in understanding and presenting the research results. All of the forest plots are graphed on a scale from -1.0 to +1.0. In this research study, 0 indicates no effect on software performance effectiveness. In the hypothesis testing, if 95% confidence interval of the difference in means included 0, then the confidence intervals of the means overlapped. Conversely, if the 95% confidence interval did not include 0, then the results were statistically significant. I summarized the study findings following these guidelines, as shown in Table 17.

The effectiveness measures for the software performance testing improvements were the moderating variables in the subgroup analysis. Table 18 shows a summary of the study findings based on the effectiveness measures. The findings are illustrated in the forest plots shown in Appendix I. The four moderating variables (the dependent variables of interest) assessed in the meta-analysis were:

- Total Defects Results Shown in Forest plot in Figure I2
- Defect Detection Rate Results Shown in Forest plot in Figure I4
- Defects By Phase Results Shown in Forest plot in Figure I6
- Total Testing Hours Results Shown in Forest plot in Figure I8

Figure I2 in Appendix I is a graphical representation of the meta-analysis results for the total defects effectiveness measure. In this forest plot, *A* denotes the subgroup of studies where DOE techniques were applied and *B* denotes the subgroup that did not have DOE techniques applied. The scale for the plot is -1.00 to +1.00 with 0.00 evenly

dividing the two subgroups. Those studies on, or closest to 0.00, had little or no effect or impact on software performance testing. The farther away from the 0.00 midpoint, the more statistically significant the effect size was in support of studies in either subgroup A or subgroup B. The overall effect size for the subgroup A studies were compared to the overall effect size for the subgroup B studies and the end result is distinguished on the plot by a diamond shape. Note that the diamond is to the left of 0.00 indicating that the meta-analysis pointed to the subgroup A, studies that had DOE techniques applied, to be more effective in software performance testing.

In Appendix I, Figure I4, the meta-analysis results for the defect detection rate measure are shown. In this Forest plot, *A* denotes the subgroup of studies where DOE techniques were applied and *B* denotes the subgroup that did not have DOE techniques applied. The scale of the graph is -1.00 to +1.00 with 0.00 evenly dividing the two subgroups. In the plot in Figure I4, the final meta-analytic result of analyzing the two subgroups of studies is denoted by the diamond shape. The diamond is to the right of 0.00 indicating that the meta-analysis resulted in subgroup B, studies that did not have DOE techniques applied, were more effective in software performance testing.

In Figure I6, in Appendix I, the defects by phase effectiveness measure meta-analytic results are depicted. In this Forest plot, *A* denotes the subgroup of studies where DOE techniques were applied and *B* denotes the subgroup that did not have DOE techniques applied. The scale of the graph is -1.00 to +1.00 with 0.00 evenly dividing the two subgroups. In the plot in Figure I6, the final meta-analytic result from the

computational analysis for the two subgroups of studies is denoted by the diamond shape to the right of 0.00 which indicates that the meta-analysis demonstrated the subgroup of studies that did not have DOE techniques applied as being more statistically effective in software performance testing.

In Appendix I, Figure I8, the meta-analysis results for the effectiveness measure, total testing hours, are shown. As in the previous forest plot labels, *A* denotes the subgroup of studies where DOE techniques were applied and *B* denotes the subgroup that did not have DOE techniques applied. The final meta-analytic result of analyzing the two subgroups of studies is denoted by the diamond shape. The diamond to the right of the 0.00 midpoint points to the subgroup B of studies that did not have DOE techniques applied as being more effective in software performance testing.

The research findings for the meta-analysis are summarized in Table 17. This table summarizes the meta-analysis results by subgroup per effectiveness measure. The corresponding graphical results are presented in Appendix Figure I2, Figure I4, Figure I6, and Figure I8 in Appendix I.

Table 17

Summary of Study Findings for Effectiveness Measures

Effectiveness Measure	Which side of 0 does the label “Favors A” (DOE) lie?	Which side of 0 does the effect size and the 95% confidence interval lie?	Meta-analysis results
Defects Detected	Left	Left	Rejected null hypothesis. DOE Subgroup (A) is more effective in software performance testing increases testing.
Defect Detection Rate	Left	Right	Failed to reject null hypothesis. DOE Subgroup (A) does not increase testing effectiveness on software performance testing.
Defects Detected by Phase	Left	Right	Failed to reject null hypothesis. Effect size for both DOE Subgroup (A) and NoDOE Subgroup (B) are right of 0. Results indicated that there is no statistical difference in means for the subgroups.
Testing Hours	Left	Left	Rejected null hypothesis. DOE Subgroup (A) is more effective in software performance testing increases testing.

Note. The term, *Favors*, is used in meta-analysis by Borenstein et al., (2009) to indicate the direction of the results.

The key finding for this research study, at the dependent variable level, was that for the *Detected Defects* and *Testing Hours* effectiveness measures it was clearly shown that there was a statistical significance for the impact applying DOE techniques has on improvements in software performance testing effectiveness. In meta-analysis, the effect

size measures, which measure impact, are absolute. In Table 17, the group representing the null hypothesis is to the right of the 0 and labeled Favors B in the plots in the appendixes. For the defect detection rate, the null hypothesis was not rejected. In Figure I4, the final overall effect size (represented by the diamond on the plots) is to the right of the '0' and in the section denoted Favors B. Similarly, for defects detected, the overall effect size is left of the zero and denoted in the section of the plot labeled Favors A or the subgroup with DOE techniques applied. For testing hours, in Figure I8, note the diamond is to the right of the '0', in the Favors B portion of the plot. However, the confidence interval for subgroup B (NoDOE) is to the left of 0 and the overall effect size is to the left of the combined effect size for subgroup A (DOE). Thus, the interpretation for this effectiveness measure is that it shows statistical significance for improvements from applying DOE techniques in software performance testing.

Summary

This chapter detailed the results from the meta-analysis conducted for this study. The research question as to whether there was a relationship between applying DOE techniques to test case design and the effectiveness of software performance testing was addressed with meta-analysis. The meta-analysis was performed using studies that applied DOE techniques and studies that did not apply DOE techniques. The findings for the research answered the question and proved that the effectiveness of software performance testing is improved when DOE techniques are applied. The study findings showed this at the subgroup level. Drilling down to the studies within the DOE subgroup, the findings also showed which of the effectiveness measures examined were influential in this testing improvement. These research findings validated the results of the isolated original studies included in this study. The results for this research study are summarized in Table 18.

Table 18

Research Results Summary

	CMA V2 Software Meta-analysis Results	Hypothesis Testing Results	<i>p</i> value	Confidence Interval
Overall	DOE subgroup is more effective in software performance testing	Rejected null hypothesis (<i>Z</i> test).	< 0.0001	(0.433 , 0.559) (-0.370 , -0.244)
Defects Detected	DOE subgroup is more effective in software performance testing	Rejected null hypothesis (<i>t</i> test).	0.0016	(-0.7051 , -0.1597)
Defect Detection Rate	DOE subgroup is not more effective in software performance testing	Did not reject null hypothesis (<i>t</i> test).	0.929	(-1.012, 0.936)
Defects Detected by Phase	DOE subgroup is more effective in software performance testing	Results suggested no statistical difference between the means for the subgroups.	-0.449	(-0.548 , 0.356)
Testing Hours	DOE subgroup is more effective in software performance testing	Rejected null hypothesis (<i>t</i> test).	0.0022	(-1.321 , -0.277)

In summary, the key finding of this study is that applying DOE techniques in the test case design of software testing has a positive effect on the software performance testing effectiveness. The hypotheses testing and the meta-analysis computational analysis showed that the statistical strength of that impact depended on the effectiveness measurement used in reporting the data in the findings.

The results from this study were significant enough to warrant recommending further study on applying DOE in software performance testing. Chapter 5 presents a detailed discussion of the research experience and the interpretation of the findings.

Chapter 5: Discussion, Conclusions, and Recommendations

Overview

The discussion in Chapter 1 through Chapter 3 set the framework for this research study by detailing the problem, exploring the literature, and defining the research methodology. I presented the data collection procedure, the data preparation, and research results in Chapter 4. In this chapter, the discussion focuses on the study findings and interpretations. Also, this chapter covers the limitations of the research conducted and the possible threats to the validity of the findings, along with recommendations for future research. In this research, I highlighted positive implications for social change. Finally, the chapter ends with the conclusions derived from conducting this research.

As discussed in Chapter 1, the fast pace of technological advances and society's reliance on that technology have caused a heightened awareness for the quality of software and for software-based products. This awareness of quality has, in turn, heightened and increased society's demand for reliable software products delivered after effective software performance testing. In the spirit of continuous improvements, consumer safety, and success in the business world, software performance improvement studies are occurring continually. Reviewing the research literature, I noted many instances of such research efforts, as evidenced by the range of studies discussed in Chapter 2. The problem, though, was that these studies seemed to be isolated efforts in the research community. The settings for the original investigations ranged from research labs in the business world, to research labs in the education arena of universities, to

efforts from the software testing industry practitioner, to joint efforts by some combination of these. For example, Bandurek (2005) asserted that applying DOE techniques to software performance testing identifies unwanted interactions between factors, something which will almost always be missed by the traditional testing methods. Bandurek went on to declare that DOE techniques not only improve efficiency and effectiveness in testing, but they can also reveal problems in the process, as well as the resulting software-based products.

The question of whether the findings from these original, seemingly isolated study instances really are valid remained open. The results of this study proved that they were valid. Noticeably missing in the literature was software performance testing improvement investigations where the focus was a collective group of various software performance testing studies validating software performance testing effectiveness. Therefore, the purpose of this research was to evaluate the reported findings from the included primary software performance testing studies synthesized as the findings from the aggregate of those studies, and add to the current testing body of knowledge.

The nature of this study was that it was an investigation across a group of original individual software performance testing studies, where each individual study reported findings showing statistically significant evidence for improvements in software performance testing effectiveness and efficiency. The research question centered on the examination of the relationship between applying DOE techniques in the test case design and software testing performance improvements. The research question was: What is the

relationship between the DOE techniques applied to test case design during testing and the effectiveness of the software performance testing? The study answered the question. Applying DOE techniques to test case design during software performance testing improves software performance testing effectiveness.

As discussed in Chapter 3, meta-analysis was the research method utilized in this study. The major criteria for including the original studies were that the findings indicated software testing performance improvements and the findings were reported in a peer-reviewed journal.

Chapter 4 contains the meta-analysis results, which answered the research question. Not only did the findings show that there is a relationship between applying DOE techniques and software performance testing effectiveness, the findings also validated the findings of the original studies included in this research.

Interpretation of Findings

When tested at the subgroup level, the findings of the first hypothesis demonstrated that the subgroup that had DOE techniques applied in the software performance testing had more impact on testing effectiveness. Hence, the conclusion is that applying DOE techniques in the test case design during software testing has a more positive impact on the resulting software performance effectiveness. The first hypothesis included any of four dependent variables in an included study. However, the results for each of the dependent variables in the four subsequent hypotheses, revealed how each of the four dependent variables contributed to the overall finding. The effectiveness

measures, detected defects and testing hours, proved to be influencers for testing improvements in software performance testing. On the other hand, for the defect detection rate and defects detected by phase, effectiveness measures had different results. The subgroup of studies that reported effectiveness in terms of defect detection rate and did not have DOE techniques applied proved better than the subgroup that did apply DOE techniques. Lastly, the defects detected by phase effectiveness measure showed that there was no statistical significance between the two subgroups.

The findings related to the number of detected defects underscores the fact that managers in the business world understand defects, all software testers understand defects, and all customers or end-users of software and software products understand defects. For software developers, software testers, and customers, the real test of the quality of the software or software product comes down to the software defects. This is a metric understood by all of these parties. All of the original studies published findings on software performance testing improvements. The DOE techniques in my research study dealt with factor covering, two-way interactions, factor combinations, and the Taguchi approach. All of the DOE techniques served to guarantee a wider and deeper coverage of the code. With more branches and code covered in the test cases designed applying such techniques, it is not surprising that the software testing performance is more effective. The DOE techniques only augmented the performance of what is already a long-established method for measuring the effectiveness of the software testing process. Hence, for defects, in meta-analysis terminology, the reporting of all the findings was

apples-to-apples. With the subgroup of studies where DOE techniques were applied having the greater effect size, the meta-analysis results served to validate this measure (total defects) for reporting software performance testing effectiveness. In software testing, reporting defects has long been the practice for measuring performance testing effectiveness. This research has shown that *defects are* still useful for measuring software testing effectiveness when DOE techniques are applied.

As shown in Table 1, each of the three hypotheses where the null hypothesis was rejected had a sample size of at least 30. The two null hypotheses that were not rejected had sample sizes of only 9 and 18. The minimum number of original studies needed for this research was calculated to be 96. Note that hypothesis one, where all 96 studies were included, met this requirement and the finding was conclusive in showing the DOE subgroup to be more effective in software performance testing. The other four hypotheses that focused on some subset of these studies had mixed results. Hence, these results indicated that sample size had a significant impact on the study findings.

As for the study findings measured in terms of defect detection rate and phase defects detected, two other factors might have influenced the outcome in this study. First, the test methodology might not have been the best fit for this research. For example, for the studies reporting findings measured by phase in which defects were detected, the phases varied. Some studies reported defects from the requirements phase through acceptance testing, others reported findings for unit testing, and still others systems integration. While all studies in this category, *defects detected by phase*, did report

findings, the findings were not for a single testing phase. The findings covered some combination of testing phases (for example, unit, integration, or system testing). For this reason, the synthesis of the meta-analysis was based on findings that could be categorized as *apples-to-oranges*. Similarly, for the studies reporting total test time, findings were reported in nanoseconds, seconds, hours, and days. Thus, for defect detection rate, phase defects detected, and total test execution time, the commonly voiced meta-analysis concern of *apple-to-oranges* possibly affected this study's findings.

Second, the measures for reporting the findings were possibly a mismatch for the DOE techniques applied. Collecting data in the same manner and using a different metric for reporting it has the potential to skew the metrics or in this research, the findings. An analogy here is an organization that collects the right data but imposes the wrong measures for metrics reporting. In such instances, was the wrong data collected or are the wrong metrics being reported? The intent of the data collected for this research was assessing software performance testing effectiveness. The measures defect detection rate, phase defects detected, and total test execution time all spoke to process, which impacted cost. So the improvements would be in process by reducing bottlenecks to increase the rate of defect detection, defects detected earlier, and reducing the time spent executing tests. Looking at the data (the studies) and reviewing the factors in Table 1 for when to use Classical DOE techniques or the Taguchi approach, perhaps more of the studies reporting these findings in these measures should have utilized the Taguchi approach, which is better at addressing process issues.

Limitations of the Research Study

This research study utilized the meta-analysis research method. The research method itself is viewed by many in the research community as a limitation on the study (Borenstein et al., 2009). While Appendix B is a testament to the large number of professionals in the research community who have not only embraced this research methodology but also the software package used in this study, there are perhaps just as many who have not embraced it. Not only have many in the research community not embraced meta-analysis, but have also been very vocal in their criticisms of the research method. As these criticisms are levied against the methodology used in this study, they can be viewed as limitations of the study. Several of these criticisms are discussed here.

Threats to Validity

Meta-analysis is most noteworthy as a disciplined technique for aggregating and synthesizing research findings (Lipsey and Wilson, 2001). Moreover, a prime reason for a researcher to conduct a meta-analytic procedure is to validate prior research findings. The criticism against the meta-analytic procedure threatens the procedure's validity. This threat in turn poses a threat against the findings of any study employing the methodology.

Threats to Generalizability

The criticism of the meta-analytic procedure, notwithstanding even for those in the research community who have embraced the methodology, the possible reluctance of industry software testing practitioners to embrace the technical methodology poses a limitation. While the application of DOE has been shown to improve the software

performance testing effectiveness, it is less common in many software testing industries. As seen in the settings of original software testing studies, most were conducted in the academic arena. For many organizations in the business world, software testing is steeped in the traditional methodologies (for example, break-it testing or stress testing). For other organizations, the use of software tools in the design and generation of test cases is seen as being on the leading edge in the use technology to improve software performance testing. This possible threat to the generalizability of this research can be summed up in thoughts by Bandurek (2005), who attributed the rigorous mathematical methods and statistical tools inherent in the DOE methodology as deterrents to the mainstream software testing communities' reluctance to embrace it. He also hinted that many industries would not apply the methodology due to certain industry regulation requirements from their customers, which encourages validation methods that are more traditional in nature, or could pass standard audit requirements, or standard certification processes.

One Number Summarization of a Research Study

Reducing research findings to a single number is another criticism offered by some in the research community. The research critics who use this as an argument against meta-analysis focus on the fact that the procedure reduces an analysis to single summary effect size statistic. They submit that doing so ignores the fact that effect size statistics may vary from research study to research study.

File Drawer Problem

In meta-analysis, *availability bias* or the *file-drawer* problem (the realization that possibly relevant literature might be yet unpublished and to discount such literature could introduce a bias in the findings) is another noted criticism. This criticism, also known by many in the research community as *publication bias*, refers to possibly missing important data. The fact that there could possibly be unpublished studies gave rise to the *file drawer problem* label. Note that this criticism is not just true for meta-analysis but could apply to any research. However, because of its association with meta-analysis, the file drawer problem poses a limitation for the validity of this study.

Mixing Apples and Oranges

The main argument for this criticism is that when researchers combine original studies, important differences could be ignored. Additionally, synthesizing studies with different characteristics that could be so totally opposite is a real concern. Combining such studies might result in a combination that invalidates the research findings. Meta-analysis brings together original studies with different characteristics. Thus, this mixing of different characteristics poses a limitation for this study.

Recommendations for Future Research

The analysis revealed several areas to address in future research. In some instances questions were raised and in others instances some facets of software performance testing could have been given more attention. As a result, several recommendations come to mind. First, any future research should make an effort for

more studies from the business world, maybe more technical papers from high technology companies to balance the world of academia. This would address any appearance of publication bias. Second, the original researchers operationalized improvements in software performance testing in a variety of ways. The validity of futures studies would benefit by sticking to a single effectiveness measure and a single testing phases. Future research should be especially mindful of the apples to oranges criticism so often made in the research community, regardless of the research method. Third, this analysis revealed little emphasis focused on the software testers. Regardless of the testing being performed, the software tester is integral to the process. More emphasis should be given to the software tester in any future research.

Software Testing Publication Bias

Publication bias is a common issue for most meta-analyses. The funnel plots generated for this meta-analysis proved that there was no difference with this one. The plots hinted at missing data. The literature reviewed for Chapter 2 was evidence that there is quite a bit of literature on findings from software performance testing studies. Future software performance testing research should make more of an effort to include a more even distribution of studies from academia, industry, and the research labs. On the other hand, restricting the meta-analysis to software performance testing studies conducted in the same test setting or arena might prevent availability bias. Hence the recommendation in this area is to select a particular community of researchers (for example, Agile Software Testing) and solicit papers from those researchers. Opening up to include

studies presented at conferences, workshops, and published by technical employees in businesses would be a way to address publication bias or file drawer syndrome. The best recommendation to address publication bias is to select a research method other than meta-analysis.

Software Testing Effectiveness Measurement

In this meta-analysis, the researchers in the original studies operationalized and reported testing performance testing effectiveness in several different ways. It made coding for this meta-analysis cumbersome, as predicted by Lipsey and Wilson (2001). While all original studies ultimately addressed the cost associated with software performance testing, with some researcher measuring effectiveness in terms of the total number of the defects detected, others were measuring effectiveness in terms of the number of test cases, and still others measuring effectiveness in terms of the total test time, the comparison of the effect sizes in the analysis can be difficult. In future research, every effort should be made to include only studies that use the same measurement for reporting software performance testing effectiveness.

Software Testers

The analysis revealed that, depending on the type of software testing, the proficiency of the software tester could be very valuable to the software testing results. The vast majority of the original investigation in this research meta-analysis focused on the design methodology of the test cases and test suites. The software testers, if included at all, seemed incidental. Upon closer analysis, the same trend was observed in the

studies discussed in Chapter 2. Future research should address this gap with emphasis on the proficiency of tester in areas such as knowledge of DOE techniques, software testing tools, and testing techniques. For example, future research could assess any trade-offs between applying a design methodology where there is rigorous mathematical or statistical framework and the benefits to be had from software testers with vast knowledge of the system under test or a long standing relationship and understanding of the customer.

Depending on the phase or type of testing, a software tester's lack of testing or software development knowledge could be more of an advantage than a hindrance. There was a noticeable gap in the number of the software performance testing investigation where the software tester was sufficiently considered. Could the same be said for software testers applying DOE techniques when there is not a proficiency in experimental design techniques? Any gain to be obtained from applying DOE techniques could easily be overshadowed by the time consumed with the upfront test case design and preparation activities. Future research could focus on investigating factors that might be constraining the effectiveness of the software tester in the software performance testing process.

Implications for Social Change

The potential impact of this research study is far-reaching, from society in general, to policy and regulations governing software performance testing, to business organizations in the software industry, to the software testing professional.

Potential Societal Impact

With today's increasing dependence on technology and the fast pace of technological change, society is more and more invested in the quality and reliability of software performance testing. From the automobiles driven, to children toys, to the mission critical software embedded in our national defense systems, society is impacted. Individuals, families, and organizations depend on automobiles. Our national defense and national policy are directly affected by performance testing effectiveness and efficiency. The original studies for this research covered academia, research labs, and private industry. Covering a cross-section of society with the original included studies showed that this research clearly impacts society. Moreover, based on the literature and my research, it is clear that improvements in software performance testing impact society at all levels.

Potential Impact for the Software Testing Industry

Software testing, as discussed throughout this research, is costly (Nirpal & Kale, 2012; Lazic & Velasevic, 2004; Nirpal & Kale, (2012). The impact of this research is significant for the software testing industry in that it provides a methodology that addresses testing costs. Reducing the size of the test suite by reducing the number of test cases that need to be executed by increasing the coverage of existing test cases, directly corresponded to the amount of time needed for the software testing process. The measurable reduction in testing time translates performance improvements into lower software testing cost, which directly impacts an organization's bottom line.

Potential Impact for the Software Testing Professionals

The potential impact for the software testing professional is the application of scientific methodology to the design of test cases. This removes some of the subjectivity that might enter test case selection, making the process much more repeatable. In the competitiveness of industry, organizations are always looking for ways to differentiate themselves to gain customers and market share. Software testing professionals with the skills to apply DOE techniques would certainly fit the bill.

On the other hand, this study revealed instances where the subjectivity of the tester can be an advantage in the testing process. The tester's relationship with the customer and familiarity with how the system is used in the customer's organization can be invaluable to the test case design and test suite selection. So, while the application of a technical methodology in the software testing process can be a differentiator, it does not diminish the importance of the software tester, as suggested by Sirathienchai, Sophatsathit, and Dechawatanapaisal (2012). The use of experimental design techniques should become more pervasive among software testers. Proficiency in the application of experimental design techniques could prove, I think after this research, an invaluable skill set for software test professionals, and thus increase the effectiveness of software performance testing.

Conclusion

While most of the research efforts since 1980 into the improvements of software testing occurred in a university setting (Watkins, 1982), there were a few studies

performed outside of academia. Some of these early studies were taking place in the business community. Though these efforts were typically research and development projects and among the first to see a reduction in funding during economic downturns, these studies were significant in many ways. They paved the way and were foundational for other research efforts. The recent rise in the research literature serves to underscore the importance of these early studies. The cost of software testing is still the focus of many of these investigations, whether in the form of reduced test execution time (Li & Song, 2008) or detecting defects as early as possible in the testing process (Baharom & Shukur, 2008). These software performance testing efforts, both those discussed in Chapter 2 and those included in the meta-analysis, described the use of testing scientific methodologies, like pairwise and combinatorial test strategies. Many of the studies referenced not only the early studies on software testing, but each other's works and soon certain researchers' names were recognizable in particular areas of software performance testing improvements investigations.

All of the original studies included in my meta-analysis were shown to positively impact software performance testing. Did the DOE techniques produce more statistical significance? The findings, as depicted in Table 18, showed that applying DOE techniques was more statically significant than those not employing DOE. The study proved this finding at the overall subgroup level, directly comparing the effectiveness of the subgroup of studies with DOE applied to the subgroup of studies where DOE was not applied. Moreover, the meta-analysis allowed the research to drill down to show exactly

which categories of effectiveness measures within the DOE subgroup were more influential in improving software performance effectiveness. Particularly, the greater statistical significance in the synthesis across all of the original software testing studies that applied DOE techniques was shown to be influenced by effectiveness measures detected defects and testing hours.

In conclusion, the findings of this study provide incentive for further study in this area. The message from this study is that there is a positive impact on software performance testing from applying DOE techniques. Applying DOE techniques improves software performance testing effectiveness. The software testing community, the software industry, and software test professionals should take note. There should continue to be more investigations in this area. For the sake of continuous improvement in software testing, studies need to continue so that more benchmarks are conducted and more companies adopt best testing practices that incorporate DOE techniques.

References

References marked with an asterisk (*) indicate studies included in the meta-analysis.

Aczel, A. D., & Sounderpandian, J. (2006). *Complete business statistics* (6th edition).

Boston, MA: McGraw-Hill.

*Ahmad, N. N., Khan, M. M., & Rafi, L. S. (2010). Software reliability modeling incorporating log-logistic testing-effort with imperfect debugging. *AIP Conference Proceedings*, 1298(1), 651–657. doi:10.1063/1.3516395

*Ahmed, B. S., & Zamli, K. Z. (2011). A review of covering arrays and their application to software testing. *Journal of Computer Science*, 7(9), 1375–1385.

<http://www.journalofcomputerscience.com/>

*Alalfi, M. H., Cordy, J. R., & Dean, T. R. (2009). Modelling methods for web application verification and testing: state of the art. *Software Testing: Verification & Reliability*, 19(4), 265–296. doi:10.1002/stvr.401

*Alshraideh, M., Mahafzah, B. A., & Al-Sharaeh, S. (2011). A multiple-population genetic algorithm for branch coverage test data generation. *Software Quality Journal*, 19(3), 489–513. doi:10.1007/s11219-010-9117-4

*Alsmadi, I. (2012). Using test case mutation to evaluate the model of the user interface. *Computer Science Journal of Moldova*, 20(1), 82–106.

<http://www.math.md/publications/csjm/>

- *Andrews, J. H., Menzies, T., & Li, F. H. (2011). Genetic algorithms for randomized unit testing. *IEEE Transactions on Software Engineering*, 37(1), 80–94.
doi:10.1109/TSE.2010.46
- Antony, J. (2006). Taguchi or classical design of experiments: A perspective from a practitioner. *Emerald Sensor Review*, 26(3), 227–230.
doi:10.1108/02602280610675519
- Antony, J., Chou, T., & Ghosh, S. (2003). Training for design of experiments. *Work Study*, 52(6/7), 341–346.
- *Askarunisa, A., Prameela, P., & Ramraj, N. (2009). A proposed agent based framework for testing data-centric applications. *International Journal of Computational Intelligence Research*, 5(4), 429–452. <http://www.ripublication.com/ijcir.htm>
- *Baharom, S., & Shukur, Z. (2008). The conceptual design of module documentation based testing tool. *Journal of Computer Science*, 4(6), 454–462.
<http://www.journalofcomputerscience.com/>
- *Baier, C., Haverkort, B. R., Hermanns, H., & Katoen, J. (2010). Performance evaluation and model checking join forces. *Communications of the ACM*, 53(9), 76–85.
doi:10.1145/1810891.1810912
- *Baluda, M., Braione, P., Denaro, G., & Pezzè, M. (2011). Enhancing structural software coverage by incrementally computing branch executability. *Software Quality Journal*, 19(4), 725–751. doi:10.1007/s11219-011-9150-y

- *Bandurek, G. R. (2005). Using design of experiments in validation. *Biopharm International*, 18(5), 40–42, 44, 46–48, 50, 52.
<http://www.biopharminternational.com/>
- Bartlett, J., Higgins, C., & Kotrlik, J. (2001). Organizational research: Determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal*, 19(1), 43–50.
- Baugh, F., & Thompson, B. (2001). Using effect sizes in social science research: New APA and journal mandates for improved methodology practices. *Journal of Research in Education*, 11(1), 120–129.
- *Belli, F., Budnik, C., & White, L. (2006). Event-based modeling, analysis and testing of user interactions: approach and case study. *Software Testing: Verification & Reliability*, 16(1), 3–32. doi:10.1002/stvr.335
- *Berling, T., & Runeson, P. (2003). Efficient evaluation of multifactor dependent system performance using fractional factorial design. *IEEE Transactions on Software Engineering*, 29(9), 769–781. <http://www.computer.org/portal/web/tse1>
- *Bida, A. S. (2009). Software testing improvement: Factors for success. *Journal of the Quality Assurance Institute*, 23(4), 4–7.
- Borenstein, M., Hedges, L., & Rothstein, H. (2007). *Introduction to meta-analysis*.
<http://www.meta-analysis.com/index.php>.
- Borenstein, M., Hedges, L., Higgins, J., & Rothstein, H. (2009). *Comprehensive meta analysis Version 2.0 tutorial*. <http://www.meta-analysis.com/index.php>.

- Borenstein, M., Hedges, L., Higgins, J., & Rothstein, H. (2009). *Introduction to comprehensive meta-analysis*. West Sussex, UK: Wiley and Sons.
- *Briand, L.C., Labiche, Y., & He, S. (2009). Automating regression test selection based on UML designs. *Information and Software Technology Journal*, 51, 16–30.
doi:10.1016/j.infsof.2008.09.010
- *Bryce, R., & Colbourn, C. J. (2006). Prioritized interaction testing for pairwise coverage with seeding and avoids. *Information and Software Technology Journal*, 48(10), 960–970. <http://www.journals.elsevier.com/information-and-software-technology>
- *Bryce, R., & Colbourn, C. J. (2007). The density algorithm for pairwise interaction testing. *Software Testing, Verification, and Reliability*, 17(3), 159–182.
[http://onlinelibrary.wiley.com/journal/10.1002/\(ISSN\)1099-1689](http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1099-1689)
- *Bryce, R. C., & Colbourn, C. J. (2009). A density-based greedy algorithm for higher strength covering arrays. *Software Testing: Verification & Reliability*, 19(1), 37–53. doi:10.1002/stvr.393
- *Bryce, R. C., Memon, A. M., & Sampath, S. (2011). Developing a single model and test prioritization strategies for event-driven software. *IEEE Transactions on Software Engineering*, 37(1), 48 <http://www.meta-analysis.com/index.php64>.
doi:10.1109/TSE.2010.12
- *Cai, K., Zhao, D., Liu, K., & Bai, C. (2007). A mathematical modeling framework for software reliability testing. *International Journal of General Systems*, 36(4), 399–463. doi:10.1080/03081070600957939

- *Cangussu, J. W., Cooper, K., & Wong, W. (2009). A segment based approach for the number of test cases for performance evaluation of components. *International Journal of Software Engineering & Knowledge Engineering*, 19(4), 481–505.
<http://www.worldscientific.com/worldscinet/ijseke>
- *Chen, T. T., Lau, M. M., Sim, K. K., & Sun, C. C. (2009). On detecting faults for Boolean expressions. *Software Quality Journal*, 17(3), 245–261.
doi:10.1007/s11219-008-9064-5
- *Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011). Using program slicing to improve the efficiency and effectiveness of cluster test selection. *International Journal of Software Engineering & Knowledge Engineering*, 21(6), 759–777.
doi: 10.1142/S0218194011005487
- Cheung, M. (2008). A model for integrating fixed-, random-, and mixed-effects meta-analyses into structural equation modeling. *Psychological Methods*, 13(3), 182–202. doi:10.1037/a0013163.
- *Ciupa, I. I., Pretschner, A. A., Oriol, M. M., Leitner, A. A., & Meyer, B. B. (2011). On the number and nature of faults found by random testing. *Software Testing: Verification & Reliability*, 21(1), 3–28. doi:10.1002/stvr.415
- *Clarke, P. J., Power, J. F., Babich, D., & King, T. M. (2012). A testing strategy for abstract classes. *Software Testing: Verification & Reliability*, 22(3), 147–169.
doi:10.1002/stvr.429

- *Cohen, D., Dalal, S., Fredman, M., & Patton, G. (1997). The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7), 437–444d. doi:14598705
- *Cohen, M., Dwyer, M., & Shi, J. (2008). Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering*, 34(5), 633–650. doi:1562470241
- Combinatorial Testing. (n.d.). *Automated Combinatorial Test for Software (ACTS)*.
- Comprehensive Meta-Analysis. (2011). (Version 2) [Computer software]. Englewood, NJ: Biostat. <http://www.meta-analysis.com/index.php>
- *Dalal, S.R., & Mallows, C.L. (1998). Factor-covering designs for testing software. *Technometrics*, 40(3), 234–243.
- DeCoster, J. (2009). *Meta-analysis notes*. <http://www.stat-help.com/notes.html>
- *Devaraj, E. E., Kumar, S. S., Kavi, T. T., & Rajani Kanth, K. K. (2011). Predicting the software performance during feasibility study. *IET Software*, 5(2), 201–215. doi:10.1049/iet-sen.2010.0075
- Dieckmann, N., Malle, B., & Bodner, T. (2009). An empirical assessment of meta-analytic practice. *Review of General Psychology*, 13(2), 101–115. doi:10.1037/a0015107
- *Dunietz, I.S., Ehrlich, W.K., Szablak, B.D., Mallows, C.L., & Iannino, A. (1997). Applying design of experiments to software testing. *ACM*, 113(5), 205–215. doi:0-89791-914-9/97/05

- Fidler, F., & Thompson, B. (2001). Computing correct confidence intervals for ANOVA fixed- and random-effects effect sizes. *Educational and Psychological Measurement*, 61, 575–604. <http://www.sagepub.com/journals/Journal200914>
- Fitzgerald, S., & Rumrill, P. (2003). Meta-analysis as a tool for understanding existing research literature. *Work*, 21(1), 97–103. doi:1051-9815/03
- *Forbes, M., Lawrence, J., Lei, Y., Kacker, R., & Kuhn, R. (2008). Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5), 287–297. <http://nvlpubs.nist.gov/nistpubs/jres/114/1/cover-image.htm>
- *Foster, G. (2005). User dyads in software testing: bypassing the need for expert observers. *British Journal of Educational Technology*, 36(2), 205–216. doi:10.1111/j.1467-8535.2005.00453.x
- *Fraser, G., & Arcuri, A. (2013). Whole test suite generation. *IEEE Transactions on Software Engineering*, 39(2), 276–291. doi:10.1109/TSE.2012.14
- Giannakopoulou, D., Bushnell, D. H., Schumann, J., Erzberger, H., & Heere, K. (2011). Formal testing for separation assurance, *Annals of Mathematics and Artificial Intelligence*, 2011. doi:10.1007/s10472-011-9224-3
- *Goel, A. A., Gupta, S. C., & Wasan, S. K. (2008). COTT -- A testability framework for object-oriented software testing. *International Journal of Computer Science*, 3(1), 44–51. <http://www.journalofcomputerscience.com/>

- *Goel, N., & Gupta, M. (2012). Testability estimation of framework based applications. *Journal Of Software Engineering & Applications*, 5(11), 841–849.
doi:10.4236/jsea.2012.511097
- *Grindal, M., Offutt, J. & Andler, S.F. (2005). Combination testing strategies: A survey. *Software Testing, Verification, and Reliability*, 15(3), 167–199.
[http://onlinelibrary.wiley.com/journal/10.1002/\(ISSN\)1099-1689](http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1099-1689)
- *Gupta, M., Gupta, R., & Tripathi, A. (2009). An exploratory case study in designing and implementing tight versus loose frameworks. *Journal of Software Engineering & Applications*, 2(3), 209–220. doi:10.4236/jsea.2009.23029
- *Gupta, A., & Jalote, P. (2008). An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing. *International Journal on Software Tools for Technology Transfer*, 10(2), 145–160.
doi:10.1007/s10009-007-0059-5
- *Gupta, A., Kapur, R., & Jha, P. C. (2008). Considering Testing Efficiency and Testing Resource Consumption Variations In Estimating Software Reliability. *International Journal Of Reliability, Quality & Safety Engineering*, 15(2), 77–91.
<http://www.worldscientific.com/worldscinet/ijrqse>
- *Halfond, W. J., Choudhary, S., & Orso, A. (2011). Improving penetration testing through static and dynamic analysis. *Software Testing: Verification & Reliability*, 21(3), 195–214. doi:10.1002/stvr.450

- *Hartman, A., & Raskin, L. (2003). Problems and algorithms for covering arrays. *Discrete Math*, 284(3), 149–156. <http://www.journals.elsevier.com/discrete-mathematics/>
- *He, Z., Staples, G., Ross, M., Court, I., & Hazzard, K. (1997). Orthogonal software testing: Taguchi methods in software unit and subsystem testing. *Logistics Information Management*, 10(5), 189–194. doi:86066675
- *Hoskins, D.S, Colburn, C.J., & Montgomery, D.C. (2005). Software performance testing using covering arrays. *ACM*, 131–136. doi:1-59593-087-6/05/0007
- *Houhamdi, Z., & Athamena, B. (2011). Structured system test suite generation process for multi-agent system. *International Journal on Computer Science & Engineering*, 3(4), 1681–1688. <http://www.enggjournals.com/ijcse/>
- *Hu, H., Jiang, C., & Cai, K. (2009). An improved approach to adaptive testing. *International Journal of Software Engineering & Knowledge Engineering*, 19(5), 679–705. <http://www.worldscientific.com/worldscinet/ijseke>
- Iacob, I., & Constantinescu, R. (2008). Testing: First step towards software quality. *Journal of Applied Quantitative Methods*, 3(3), 241–253. <http://www.jaqm.ro/>
- IEEE-Std 610.12. (1990). *The IEEE standard glossary of software engineering terminology*. Piscataway, NJ: IEEE Press.
- *Inoue, S., & Yamada, S. (2011). Software reliability growth modeling frameworks with change of testing-environment. *International Journal of Reliability, Quality &*

Safety Engineering, 18(4), 365–376.

<http://www.worldscientific.com/worldscinet/ijrqse>

- *Itkonen, J., Mäntylä, M. V., & Lassenius, C. (2013). The role of the tester's knowledge in exploratory software testing. *IEEE Transactions on Software Engineering*, 39(5), 707–724. doi:10.1109/TSE.2012.55
- *Just, R., & Schweiggert, F. (2011). Automating unit and integration testing with partial oracles. *Software Quality Journal*, 19(4), 753–769. doi:10.1007/s11219-011-9151-x
- *Kadry, S. (2011). A new proposed technique to improve software regression testing cost. *International Journal of Security & Its Applications*, 5(3), 45–58.
- *Kadry, S., & Kalakech, A. (2011). Cost-effectiveness of regression testing: problem and solution. *International Review on Computers & Software*, 6(3), 328–335.
- *Kaminski, G., & Ammann, P. (2011). Reducing logic test set size while preserving fault detection. *Software Testing: Verification & Reliability*, 21(3), 155–193.
doi:10.1002/stvr.442
- *Kelly, D., Thorsteinson, S., & Hook, D. (2011). Scientific software testing: Analysis with four dimensions. *IEEE Software*, 28(3), 84–90. doi:10.1109/MS.2010.88
- *Khan, M. (2010). Different forms of software testing techniques for finding errors. *International Journal of Computer Science Issues (IJCSI)*, 7(3), 11–16.
<http://ijcsi.org/>

- *Khoshgoftaar, T. M., & Szabo, R. M. (2006). Dynamic models for testing based on time series analysis. *International Journal Of Reliability, Quality & Safety Engineering*, 13(6), 581–597. <http://www.worldscientific.com/worldscinet/ijrqse>
- *Kim, J., Sung, D., & Hong, J. (2011). Performance testing of composite web-service with aspect-based WS-BPEL extension. *KSII Transactions on Internet & Information Systems*, 5(10), 1841–1860. doi:10.3837/tiis.2011.10.010
- *Klaib, M., Muthuraman, S., Ahmad, N., & Sidek, R. (2010). Tree based test case generation and cost calculation strategy for uniform parametric pairwise testing. *Journal of Computer Science*, 6(5), 542–547. <http://thescipub.com/journals/jcs>
- *Kuhn, R., Wallace, D., & Gallo, A. (2004). Software fault interactions and implications for software testing. *IEEE IT Professional*, 30(6), 418–421.
<http://www.nist.gov/index.html>.
- *Kuhn, R., Kacker, R., Lei, Y., & Hunter, J. (2009). Combinatorial software testing. *IEEE Computer Society*, 42(8), 94–96. <http://www.nist.gov/index.html>.
- *Kuhn, R., Kacker, R., & Lei, Y. (2008). Automated combinatorial test methods: Beyond pairwise testing. *Crosstalk Journal of Defense Software Engineering*, 21(6), 94–96. <http://www.nist.gov/index.html>.
- *Kuhn, R., Lei, Y., & Kacker, R. (2008). Practical combinatorial testing: Beyond pairwise. *IEEE Computer Society*, 10(3), 19–23.
<http://www.nist.gov/index.html>.

- *Kuhn, R., Kacker, R., & Lei, Y. (2009). *Random vs combinatorial methods for discrete event simulation of a grid computer network*. National Institute of Standard and Technology. <http://www.nist.gov/index.html>.
- *Kuhn, D.R., & Reilly, M.J. (2002). *An investigation of the applicability of design of experiments to software testing proceeding*. 27th NASA/IEEE Software Engineering Workshop. <http://www.nist.gov/index.html>.
- *Kumar, M., Sharma, A., & Kumar, R. (2011). Towards multi-faceted test cases optimization. *Journal Of Software Engineering & Applications*, 4(9), 550–557. doi:10.4236/jsea.2011.49064
- *Kundu, D., Sarma, M., Samanta, D., & Mall, R. (2009). System testing for object-oriented systems with test case prioritization. *Software Testing: Verification & Reliability*, 19(4), 297–333. doi:10.1002/stvr.407
- *Lazić, L. (2010). Software testing optimization by advanced quantitative defect management. *Computer Science & Information Systems*, 7(3), 459–487. doi:10.2298/CSIS090923008L
- *Lazić, L., & Velašević, D. D. (2004). Applying simulation and design of experiments to the embedded software testing process. *Journal of Software Testing: Verification & Reliability*, 14(4), 257–282. doi:10.1002/stvr.299
- *Lei, Y., Carver, R., & Kung, D. (2007). A combinatorial strategy for testing concurrent programs. *Journal of Software Testing, Verification, and Reliability*, 17(4), 207–225. [http://onlinelibrary.wiley.com/journal/10.1002/\(ISSN\)1099-1689](http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1099-1689)

- *Li, Y., & Song, Y. (2008). An adaptive and trustworthy software testing framework on the grid. *Journal Of Supercomputing*, 46(2), 124–138. doi:10.1007/s11227-007-0160-2
- Lipsey, M., & Wilson, D. (2001). *Practical meta-analysis*. Thousand Oaks, CA: Sage Publications.
- *Lun, L., Chi, X., & Ding, X. (2012). Edge coverage analysis for software architecture testing. *Journal of Software (1796217X)*, 7(5), 1121–1128.
doi:10.4304/jsw.7.5.1121-1128
- *Maheswari, B., & Valli, S. S. (2011). Algorithms for the detection of defects in GUI applications. *Journal of Computer Science*, 7(9), 1343–1352.
<http://thescipub.com/journals/jcs>
- *Mala, D., Mohan, V. V., & Kamalapriya, M. M. (2010). Automated software test optimisation framework - an artificial bee colony optimisation-based approach. *IET Software*, 4(5), 334–348. doi:10.1049/iet-sen.2009.0079
- *Marchetto, A., Ricca, F., & Tonella, P. (2008). A case study-based comparison of web testing techniques applied to AJAX web applications. *International Journal on Software Tools for Technology Transfer*, 10(6), 477–492. doi:10.1007/s10009-008-0086-x
- *Marinescu, P. D., & Candea, G. (2011). Efficient testing of recovery code using fault Injection. *ACM Transactions on Computer Systems*, 29(4), 1–38.
doi:10.1145/2063509.2063511

- *Mei, H., Hao, D., Zhang, L., Zhang, L., Zhou, J., & Rothermel, G. (2012). A static approach to prioritizing JUnit test cases. *IEEE Transactions on Software Engineering*, 38(6), 1258–1275. doi:10.1109/TSE.2011.106
- *Misirli, A., Bener, A., & Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, 19(3), 515–536. doi:10.1007/s11219-010-9128-1
- *Montanez, C., Kuhn, D.R., Brady, M., Rivello, R., Reyes, J., & Powers, M.K., (2011). An application of combinatorial methods to conformance testing for document object model events. NIST-IR 7773. 25 Jan 2011.
<http://csrs.nist.gov/groups/SNS/acts/documents/DOM-IR-nnnn-110125.doc>.
- Montgomery, D. (2009). *Design and analysis of experiments*. (7th edition) UK: John Wiley & Sons, Incorporated.
- *Mouchawrab, S., Briand, L. C., Labiche, Y., & Di Penta, M. (2011). Assessing, comparing, and combining state machine-based testing and structural testing: A series of experiments. *IEEE Transactions on Software Engineering*, 37(2), 161–187. doi:10.1109/TSE.2010.32
- Newman, D., Jacobs, R., & Bartram, D. (2007). Choosing the best method for local validity estimation: Relative accuracy of meta-analysis versus a local study versus Bayes-analysis. *Journal of Applied Psychology*, 92(5), 1394–1413.
doi:10.1037/0021-9010.92.5.1394

- *Nirpal, P. B., & Kale, K. V. (2012). Genetic algorithm based software testing specifically structural testing for software reliability enhancement. *International Journal of Computational Intelligence Techniques*, 3(1), 60–64. <http://journal-index.org/index.php/asi/article/view/117>
- *Parsa, S., & Khalilian, A. (2010). On the optimization approach towards test suite minimization. *International Journal of Software Engineering & Its Applications*, 4(1), 15–28. <http://www.sersc.org/journals/IJSEIA/>
- Peters, J., & Mengersen, K. (2008). Meta-analysis of repeated measures study designs. *Journal of Evaluation in Clinical Practice*, 14(5), 941–950. doi:10.1111/j.1365-2753.2008.01010.x
- *Pocatilu, P., & Ciurea, C. (2009). Collaborative systems testing. *Journal of Applied Quantitative Methods*, 4(3), 394–405. <http://www.jaqm.ro/>
- *Poon, P., Tse, T. T., Tang, S., & Kuo, F. (2011). Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing. *Software Quality Journal*, 19(1), 141–163. doi:10.1007/s11219-010-9109-4
- *Poulding, S., & Clark, J. A. (2010). Efficient software verification: statistical testing using automated search. *IEEE Transactions On Software Engineering*, 36(6), 763–777. doi:10.1109/TSE.2010.24
- *Prakash, V. V., SenthilAnand, N. N., & Bhavani, R. R. (2012). Agile-Fall process flow model - A right candidate for implementation in software development and testing

processes for software organizations. *International Journal of Computer Science Issues (IJCSI)*, 9(3), 457–461. <http://ijcsi.org/>

- *Rao, K., & Sastri, A. (2011). Overcoming testing challenges in project life cycle using risk based validation approach. *International Journal on Computer Science & Engineering*, 3(3), 1232–1239. <http://www.enggjournals.com/ijcse/>
- *Raske, T. (1994). More efficient software testing through the application of design of experiments. *IEEE*, 318–322. doi:10719458/94
- *Robinson, B., & White, L. (2012). On the testing of user-configurable software systems using firewalls. *Software Testing: Verification & Reliability*, 22(1), 3–31.
doi:10.1002/stvr.428
- *Sagarna, R., & Lozano, J. (2005). On the performance of estimation of distribution algorithms applied to software testing. *Applied Artificial Intelligence*, 19(5), 457–489. doi:10.1080/08839510590917861
- *Shahbazi, A., Tappenden, A. F., & Miller, J. (2013). Centroidal voronoi tessellations—A new approach to random testing. *IEEE Transactions on Software Engineering*, 39(2), 163–183. doi:10.1109/TSE.2012.18
- Singleton, R. A., & Straits, B. C. (2010). *Approaches to social research* (5th edition). NY: Oxford University Press.
- *Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012). Simulation-based evaluation for the impact of personnel capability on software testing performance.

Journal of Software Engineering & Applications, 5(8), 545–559.

doi:10.4236/jsea.2012.58063

*Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012). Using test employee capability maturity model for supporting gaps bridging in software testing.

Journal of Software Engineering & Applications, 5(6), 417–428.

doi:10.4236/jsea.2012.56048

Sjoberg, D.I.K, Hannay, J.E, Hansen, O., Kampene, V., Karahasanovic, A., Liborg, N., & Rekdal A.C. (2005). A survey of controlled experiments in software engineering.

IEEE Transactions on Software Engineering, 31(9), 733–753.

<http://www.computer.org/portal/web/tse>

Sterne, J., & Harbord, R. (2004). Funnel plots in meta-analysis. *The Stata Journal*, 4(2),

127–141. <http://www.stata-journal.com/>

Swanson, R. A., & Holton, E. F. (2005). *Research in organizations: Foundation and methods inquiry*. NY: Berrett and Koehler Publishers.

Tai, K.C., & Lie, Y. (2002). A test generation strategy for pairwise testing. *IEEE*

Transaction on Software Engineering, 28(1), 109–111.

<http://www.computer.org/portal/web/tse>

*Teasley, B. E., Leventhal, L., Mynatt, C. R., & Rohlman, D. S. (1994). Why software testing is sometimes ineffective: Two applied studies of positive test strategy.

Journal of Applied Psychology, 79(1), 142–154.

<http://www.apa.org/pubs/journals/apl/index.aspx>

- Testing Arrays. (n.d.). *Covering Arrays*. <http://math.nist.gov/coveringarrays>.
- Testing the quality of software characteristics of software. (2008). *Journal of the Quality Assurance Institute*, 22(3), 19–21.
- Thompson, B. (2001). Significance, effect sizes, stepwise methods, and other issues: Strong arguments move the field. *Journal of Experimental Education*, 70, 80–93. http://www.researchgate.net/journal/0022-0973_The_Journal_of_Experimental_Education
- Thompson, B. (2007). Effect sizes, confidence intervals, and confidence intervals for effect sizes. *Psychology in the Schools*, 44, 423–432. [http://onlinelibrary.wiley.com/journal/10.1002/\(ISSN\)1520-6807](http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1520-6807)
- Thompson, B. (2002). What future quantitative social science research could look like: Confidence intervals for effect sizes. *Educational Researcher*, 31(3), 24–31. <http://edr.sagepub.com/>
- Vacha-Haase, T., & Thompson, B. (2004). How to estimate and interpret various effect sizes. *Journal of Counseling Psychology*, 51, 473–481.
- *Watkins, M. L. (1982). A technique for testing command and control software. *Communications of The ACM*, 25(4), 228–232. <http://www.acm.org/>
- Whyte, G., & Mulder, D. (2011). Mitigating the impact of software test constraints on software testing effectiveness. *Electronic Journal of Information Systems Evaluation*, 14(2), 254–270. <http://ejise.com/main.html>

- *Yang, L., Dang, Z., & Fischer, T. (2011). Information gain of black-box testing. *Formal Aspects Of Computing*, 23(4), 513–539. doi:10.1007/s00165-011-0175-6
- *Ye, D. (2011). Automated testing framework for ODBC driver. *Journal Of Software Engineering & Applications*, 4(12), 688–699. doi:10.4236/jsea.2011.412081
- *Yoon, H., & Choi, B. (2011). A test case prioritization based on degree of risk exposure and its empirical study. *International Journal of Software Engineering & Knowledge Engineering*, 21(2), 191–209. doi:10.1142/S0218194011005220
- *Yoon, M., Lee, E., Song, M., & Choi, B. (2012). A test case prioritization through correlation of requirement and risk. *Journal of Software Engineering & Applications*, 5(10), 823–835. doi:10.4236/jsea.2012.510095
- *Yuan, X., Cohen, M. B., & Memon, A. M. (2011). GUI interaction testing: Incorporating event context. *IEEE Transactions On Software Engineering*, 37(4), 559–574. doi:10.1109/TSE.2010.50
- *Zhang, Z., & Zhou, Y. (2007). A fuzzy logic based approach for software testing. *International Journal of Pattern Recognition & Artificial Intelligence*, 21(4), 709–722. <http://www.worldscientific.com/worldscinet/ijprai>
- *Zheng, Q., & Dan-ping, W. (2009). Software testing effectiveness modeling-based on complete or missing detection data. *Journal of Communication & Computer*, 6(4), 14–19. <http://www.ijccts.org/>

- *Zielińska, A. (2012). Framework for Extensible Application Testing. *Journal of Software Engineering & Applications*, 5(5), 351–363.
doi:10.4236/jsea.2012.55041

Appendix A: Statement to the Validation of CMA Version 2

The Comprehensive Meta-Analysis, version 2 , computer software for meta-analysis was developed by a team of experts from the United States and the United Kingdom. The following is an email testament as to the quality of the software package from one of the developers.

From: Michael Borenstein [biostat100@gmail.com]
Sent: Tuesday, October 02, 2012 4:04 PM
To: gloria.johnson@waldenu.edu
Subject: Comprehensive meta-analysis

The program was tested extensively against Revman and the stata macros, which had been seen as the gold standard.

The validation data were sent to NIH as part of the reports, since the program development was funded by NIH.

This is the most widely used program in the world for meta-analysis with over 10,000 users in 50+ countries.

The algorithms are discussed in the book Introduction to Meta-Analysis (Borenstein et al) .

There are some 200 publications listed in PubMed that are based on this program.

Hope this helps
Michael

Appendix B: Comprehensive Meta-Analysis Version 2 Testimonials

"Thank you very much for the wonderful workshop at Kent State University. I really enjoyed it. I particularly like the way you organize the course, starting with the concept, then applications and examples, and finally common mistakes." Jingzhen (Ginger) Yang, PhD, MPH, Associate Professor, Department of Social and Behavioral Sciences, College of Public Health, Kent State University.

"The meta-analysis seminar was extremely clear, informative, and helpful. We were especially pleased that it was at an appropriate level for the faculty and researchers who were from various areas of specialization in health and medical sciences at our University of Medicine and Dentistry of New Jersey. Thank you." Syed S. Haque, Ph. D., Professor and Chairman, Department of Health Informatics, Director of Graduate Programs in Biomedical Informatics.

"We perform a variety of meta-analyses for academic, regulatory, and international clients. Each presents a different set of challenges regarding study design and outcome measurement. We have found CMA to be invaluable in this work. The ability of the software to capture a variety of data elements (study design, multiple outcomes, covariates/confounders) and present details of computations is important in the credibility of our work. The ease of use and ability to produce graphics in a variety of formats aids in preparation of the report. In many instances, we are required to replicate the results of CMA in another package (for example, SAS). We have always found the support staff at CMA very helpful in these replications and the results of CMA have been

replicated in every instance. CMA is a great tool in the scientific credibility of our meta-analytic studies. “ Donna F. Stroup, PhD, MSc, Data for Solutions, Inc.

“Comprehensive Meta-Analysis is an indispensable tool for efficient problem solving in meta-analyses. Regardless of whether or not you are a statistician, the software leads you to the world of meta-analysis quickly. Comprehensive Meta-Analysis is extremely easy to use and understand and it is a terrific product. “Dr. Takeharu Yamanaka, Cancer Biostatistics Laboratory, National Kyushu Cancer Center, Japan

“I have been using Comprehensive Meta-Analysis for more than 3 years and have finished a dozen meta-analysis with this software. The biggest advantage is easy to perform and manage the analysis. Studies can be added or removed from the analysis without modifying the data. There are a variety of effect size measures, including treatment difference, odds ratios, rate differences, correlations, etc. The diagnosis and transformation of the effect size is just one click away. The high-quality forest plot and comprehensive meta-regression distinguished this software from others. “ Rong Zhou, PhD, Senior Biostatistician, Medpace. Cincinnati, Ohio

Appendix C: Included Original Studies Without DOE Techniques

Table C1

References Without Design of Experiments

No.	Reference
1	Ahmad, N. N., Khan, M. M., & Rafi, L. S. (2010). Software reliability modeling incorporating log-logistic testing-effort with imperfect debugging. <i>AIP Conference Proceedings</i> , 1298(1), 651-657. doi:10.1063/1.3516395
2	Alalfi, M. H., Cordy, J. R., & Dean, T. R. (2009). Modelling methods for web application verification and testing: state of the art. <i>Software Testing: Verification & Reliability</i> , 19(4), 265-296. doi:10.1002/stvr.401
3	Alsmadi, I. (2012). Using test case mutation to evaluate the model of the user interface. <i>Computer Science Journal Of Moldova</i> , 20(1), 82-106.
4	Andrews, J. H., Menzies, T., & Li, F. H. (2011). Genetic algorithms for randomized unit testing. <i>IEEE Transactions On Software Engineering</i> , 37(1), 80-94. doi:10.1109/TSE.2010.46
5	Askarunisa, A., Prameela, P., & Ramraj, N. (2009). A proposed agent based framework for testing data-centric applications. <i>International Journal of Computational Intelligence Research</i> , 5(4), 429-452. Retrieved from Computers & Applied Sciences Complete database.
6	Baharom, S., & Shukur, Z. (2008). The conceptual design of module documentation based testing tool. <i>Journal Of Computer Science</i> , 4(6), 454-462.
7	Briand, L.C., Labiche, Y., & He, S. (2009). Automating regression test selection based on UML designs. <i>Information and Software Technology Journal</i> , 51, 16-30. doi:10.1016/j.infsof.2008.09.010
8	Bryce, R. C., Memon, A. M., & Sampath, S. (2011). Developing a single model and test prioritization strategies for event-driven software. <i>IEEE Transactions On Software Engineering</i> , 37(1), 48-64. doi:10.1109/TSE.2010.12

(table continues)

No.	Reference
9	Chen, T. T., Lau, M. M., Sim, K. K., & Sun, C. C. (2009). On detecting faults for Boolean expressions. <i>Software Quality Journal</i> , 17(3), 245-261. doi:10.1007/s11219-008-9064-5
10	Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011). Using program slicing to improve the efficiency and effectiveness of cluster test selection. <i>International Journal Of Software Engineering & Knowledge Engineering</i> , 21(6), 759-777.
11	Ciupa, I. I., Pretschner, A. A., Oriol, M. M., Leitner, A. A., & Meyer, B. B. (2011). On the number and nature of faults found by random testing. <i>Software Testing: Verification & Reliability</i> , 21(1), 3-28. doi:10.1002/stvr.415
12	Clarke, P. J., Power, J. F., Babich, D., & King, T. M. (2012). A testing strategy for abstract classes. <i>Software Testing: Verification & Reliability</i> , 22(3), 147-169. doi:10.1002/stvr.429
13	Foster, G. (2005). User dyads in software testing: bypassing the need for expert observers. <i>British Journal Of Educational Technology</i> , 36(2), 205-216. doi:10.1111/j.1467-8535.2005.00453.x
14	Fraser, G., & Arcuri, A. (2013). Whole test suite generation. <i>IEEE Transactions On Software Engineering</i> , 39(2), 276-291. doi:10.1109/TSE.2012.14
15	Goel, A. A., Gupta, S. C., & Wasan, S. K. (2008). COTT -- A testability framework for object-oriented software testing. <i>International Journal Of Computer Science</i> , 3(1), 44-51.
16	Goel, N., & Gupta, M. (2012). Testability estimation of framework based applications. <i>Journal Of Software Engineering & Applications</i> , 5(11), 841-849. doi:10.4236/jsea.2012.511097
17	Halfond, W. J., Choudhary, S., & Orso, A. (2011). Improving penetration testing through static and dynamic analysis. <i>Software Testing: Verification & Reliability</i> , 21(3), 195-214. doi:10.1002/stvr.450
18	Hu, H., Jiang, C., & Cai, K. (2009). An improved approach to adaptive testing. <i>International Journal Of Software Engineering & Knowledge Engineering</i> , 19(5), 679-705.
19	Itkonen, J., Mäntylä, M. V., & Lassenius, C. (2013). The role of the tester's knowledge in exploratory software testing. <i>IEEE Transactions On Software Engineering</i> , 39(5), 707-724. doi:10.1109/TSE.2012.55

(table continues)

No.	Reference
20	Just, R., & Schweiggert, F. (2011). Automating unit and integration testing with partial oracles. <i>Software Quality Journal</i> , 19(4), 753-769. doi:10.1007/s11219-011-9151-x
21	Kaminski, G., & Ammann, P. (2011). Reducing logic test set size while preserving fault detection. <i>Software Testing: Verification & Reliability</i> , 21(3), 155-193. doi:10.1002/stvr.442
22	Khan, M. (2010). Different forms of software testing techniques for finding errors. <i>International Journal of Computer Science Issues (IJCSI)</i> , 7(3), 11-16. Retrieved from Computers & Applied Sciences Complete database.
23	Khoshgoftaar, T. M., & Szabo, R. M. (2006). Dynamic models for testing based on time series analysis. <i>International Journal Of Reliability, Quality & Safety Engineering</i> , 13(6), 581-597.
24	Kumar, M., Sharma, A., & Kumar, R. (2011). Towards multi-faceted test cases optimization. <i>Journal Of Software Engineering & Applications</i> , 4(9), 550-557. doi:10.4236/jsea.2011.49064
25	Kundu, D., Sarma, M., Samanta, D., & Mall, R. (2009). System testing for object-oriented systems with test case prioritization. <i>Software Testing: Verification & Reliability</i> , 19(4), 297-333. doi:10.1002/stvr.407
26	Li, Y., & Song, Y. (2008). An adaptive and trustworthy software testing framework on the grid. <i>Journal Of Supercomputing</i> , 46(2), 124-138. doi:10.1007/s11227-007-0160-2
27	Maheswari, B., & Valli, S. S. (2011). Algorithms for the detection of defects in GUI applications. <i>Journal Of Computer Science</i> , 7(9), 1343-1352.
28	Marchetto, A., Ricca, F., & Tonella, P. (2008). A case study-based comparison of web testing techniques applied to AJAX web applications. <i>International Journal On Software Tools For Technology Transfer</i> , 10(6), 477-492. doi:10.1007/s10009-008-0086-x
29	Marinescu, P. D., & Candea, G. (2011). Efficient testing of recovery code using fault Injection. <i>ACM Transactions On Computer Systems</i> , 29(4), 1-38. doi:10.1145/2063509.2063511

(table continues)

No.	Reference
30	Mei, H., Hao, D., Zhang, L., Zhang, L., Zhou, J., & Rothermel, G. (2012). A static approach to prioritizing JUnit test cases. <i>IEEE Transactions On Software Engineering</i> , 38(6), 1258-1275. doi:10.1109/TSE.2011.106
31	Misirli, A., Bener, A., & Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. <i>Software Quality Journal</i> , 19(3), 515-536. doi:10.1007/s11219-010-9128-1
32	Poon, P., Tse, T. T., Tang, S., & Kuo, F. (2011). Contributions of tester experience and a checklist guideline to the identification of categories and choices for software testing. <i>Software Quality Journal</i> , 19(1), 141-163. doi:10.1007/s11219-010-9109-4
33	Poulding, S., & Clark, J. A. (2010). Efficient software verification: statistical testing using automated search. <i>IEEE Transactions On Software Engineering</i> , 36(6), 763-777. doi:10.1109/TSE.2010.24
34	Prakash, V. V., SenthilAnand, N. N., & Bhavani, R. R. (2012). Agile-Fall process flow model - A right candidate for implementation in software development and testing processes for software organizations. <i>International Journal of Computer Science Issues (IJCSI)</i> , 9(3), 457-461. Retrieved from Computers & Applied Sciences Complete database.
35	Rao, K., & Sastri, A. (2011). Overcoming testing challenges in project life cycle using risk based validation approach. <i>International Journal on Computer Science & Engineering</i> , 3(3), 1232-1239. Retrieved from Computers & Applied Sciences Complete database.
36	Robinson, B., & White, L. (2012). On the testing of user-configurable software systems using firewalls. <i>Software Testing: Verification & Reliability</i> , 22(1), 3-31. doi:10.1002/stvr.428
37	Shahbazi, A., Tappenden, A. F., & Miller, J. (2013). Centroidal voronoi tessellations—A new approach to random testing. <i>IEEE Transactions On Software Engineering</i> , 39(2), 163-183. doi:10.1109/TSE.2012.18
38	Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012). Simulation-based evaluation for the impact of personnel capability on software testing performance. <i>Journal Of Software Engineering & Applications</i> , 5(8), 545-559. doi:10.4236/jsea.2012.58063

(table continues)

No.	Reference
39	Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012). Using test employee capability maturity model for supporting gaps bridging in software testing. <i>Journal Of Software Engineering & Applications</i> , 5(6), 417-428. doi:10.4236/jsea.2012.56048
40	Teasley, B. E., Leventhal, L., Mynatt, C. R., & Rohlman, D. S. (1994). Why software testing is sometimes ineffective: Two applied studies of positive test strategy. <i>Journal Of Applied Psychology</i> , 79(1), 142-154.
41	Watkins, M. L. (1982). A technique for testing command and control software. <i>Communications Of The ACM</i> , 25(4), 228-232.
42	Yang, L., Dang, Z., & Fischer, T. (2011). Information gain of black-box testing. <i>Formal Aspects Of Computing</i> , 23(4), 513-539. doi:10.1007/s00165-011-0175-6
43	Ye, D. (2011). Automated testing framework for ODBC driver. <i>Journal Of Software Engineering & Applications</i> , 4(12), 688-699. doi:10.4236/jsea.2011.412081
44	Yoon, H., & Choi, B. (2011). A test case prioritization based on degree of risk exposure and its empirical study. <i>International Journal Of Software Engineering & Knowledge Engineering</i> , 21(2), 191-209.
45	Yoon, M., Lee, E., Song, M., & Choi, B. (2012). A test case prioritization through correlation of requirement and risk. <i>Journal Of Software Engineering & Applications</i> , 5(10), 823-835. doi:10.4236/jsea.2012.510095
46	Yuan, X., Cohen, M. B., & Memon, A. M. (2011). GUI interaction testing: Incorporating event context. <i>IEEE Transactions On Software Engineering</i> , 37(4), 559-574. doi:10.1109/TSE.2010.50
47	Zhang, Z., & Zhou, Y. (2007). A fuzzy logic based approach for software testing. <i>International Journal Of Pattern Recognition & Artificial Intelligence</i> , 21(4), 709-722.
48	Zielińska, A. (2012). Framework for Extensible Application Testing. <i>Journal Of Software Engineering & Applications</i> , 5(5), 351-363.

Appendix D: Included Studies With DOE Techniques

Table D1

References With Design of Experiments

No.	Reference
1	Ahmed, B. S., & Zamli, K. Z. (2011). A review of covering arrays and their application to software testing. <i>Journal Of Computer Science</i> , 7(9), 1375-1385. Retrieved from Computers & Applied Sciences Complete database.
2	Alshraideh, M., Mahafzah, B. A., & Al-Sharaeh, S. (2011). A multiple-population genetic algorithm for branch coverage test data generation. <i>Software Quality Journal</i> , 19(3), 489-513. doi:10.1007/s11219-010-9117-4
3	Baier, C., Haverkort, B. R., Hermanns, H., & Katoen, J. (2010). Performance evaluation and model checking join forces. <i>Communications of The ACM</i> , 53(9), 76-85. doi:10.1145/1810891.1810912
4	Baluda, M., Braione, P., Denaro, G., & Pezzè, M. (2011). Enhancing structural software coverage by incrementally computing branch executability. <i>Software Quality Journal</i> , 19(4), 725-751. doi:10.1007/s11219-011-9150-y
5	Bandurek, G. R. (2005). Using design of experiments in validation. <i>Biopharm International</i> , 18(5), 40-42, 44, 46-48, 50, 52. doi:845094131
6	Belli, F., Budnik, C., & White, L. (2006). Event-based modeling, analysis and testing of user interactions: approach and case study. <i>Software Testing: Verification & Reliability</i> , 16(1), 3-32. doi:10.1002/stvr.335
7	Berling, T., & Runeson, P. (2003). Efficient evaluation of multifactor dependent system performance using fractional factorial design. <i>IEEE Transactions on Software Engineering</i> , 29(9), 769-781. doi:431853251
8	Bida, A. S. (2009). Software testing improvement: Factors for success. <i>Journal of The Quality Assurance Institute</i> , 23(4), 4-7. Retrieved from Computers & Applied Sciences Complete database.

(tables continues)

No.	Reference
9	Bryce, R., & Colbourn, C. J. (2006). Prioritized interaction testing for pairwise coverage with seeding and avoids. <i>Information and Software Technology Journal</i> , 48(10), 960–970. Retrieved from Computers & Applied Sciences Complete database.
10	Bryce, R., & Colbourn, C. J. (2007). The density algorithm for pairwise interaction testing. <i>Software Testing, Verification, and Reliability</i> , 17(3), 159–182. Retrieved from Computers & Applied Sciences Complete database.
11	Bryce, R. C., & Colbourn, C. J. (2009). A density-based greedy algorithm for higher strength covering arrays. <i>Software Testing: Verification & Reliability</i> , 19(1), 37–53. doi:10.1002/stvr.393
12	Cai, K., Zhao, D., Liu, K., & Bai, C. (2007). A mathematical modeling framework for software reliability testing. <i>International Journal Of General Systems</i> , 36(4), 399-463. doi:10.1080/03081070600957939
13	Cangussu, J. W., Cooper, K., & Wong, W. (2009). A segment based approach for the number of test cases for performance evaluation of components. <i>International Journal of Software Engineering & Knowledge Engineering</i> , 19(4), 481-505. Retrieved from Computers & Applied Sciences Complete database.
14	Cohen, D., Dalal, S., Fredman, M., & Patton, G. (1997). The AETG system: An approach to testing based on combinatorial design. <i>IEEE Transactions on Software Engineering</i> , 23(7), 437–444d. doi:14598705
15	Cohen, M., Dwyer, M., & Shi, J. (2008). Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. <i>IEEE Transactions on Software Engineering</i> , 34(5), 633–650. doi:1562470241
17	Devaraj, E. E., Kumar, S. S., Kavi, T. T., & Rajani Kanth, K. K. (2011). Predicting the software performance during feasibility study. <i>IET Software</i> , 5(2), 201–215. doi:10.1049/iet-sen.2010.0075

(table continues)

No.	Reference
18	Dunietz, I.S., Ehrlich, W.K., Szablak, B.D., Mallows, C.L., & Iannino, A. (1997). Applying design of experiments to software testing. <i>ACM</i> , 113(5), 205–215.
19	Forbes, M., Lawrence, J., Lei, Y., Kacker, R., & Kuhn, R. (2008). Refining the in-parameter-order strategy for constructing covering arrays. <i>Journal of Research of the National Institute of Standards and Technology</i> , 113(5), 287–297. Retrieved March 27, 2011, from ABI/INFORM Global.
20	Grindal, M., Offutt, J. & Andler, S.F. (2005). Combination testing strategies: A survey. <i>Software Testing, Verification, and Reliability</i> , 15(3), 167–199. Retrieved from Computers & Applied Sciences Complete database.
21	Gupta, M., Gupta, R., & Tripathi, A. (2009). An exploratory case study in designing and implementing tight versus loose frameworks. <i>Journal of Software Engineering & Applications</i> , 2(3), 209–220. doi:10.4236/jsea.2009.23029
22	Gupta, A., & Jalote, P. (2008). An approach for experimentally evaluating effectiveness and efficiency of coverage criteria for software testing. <i>International Journal on Software Tools for Technology Transfer</i> , 10(2), 145–160. doi:10.1007/s10009-007-0059-5
23	Gupta, A., Kapur, R., & Jha, P. C. (2008). Considering testing efficiency and testing resource consumption variations in estimating software reliability. <i>International Journal Of Reliability, Quality & Safety Engineering</i> , 15(2), 77-91. Retrieved from Computers & Applied Sciences Complete database.
24	Hartman, A., & Raskin, L. (2003). Problems and algorithms for covering arrays. <i>Discrete Math</i> , 284(3), 149–156. Retrieved from Computers & Applied Sciences Complete database.
25	He, Z., Staples, G., Ross, M., Court, I., & Hazzard, K. (1997). Orthogonal software testing: Taguchi methods in software unit and subsystem testing. <i>Logistics Information Management</i> , 10(5), 189–194. doi:86066675
26	Hoskins, D.S, Colburn, C.J., & Montgomery, D.C. (2005). Software performance testing using covering arrays. <i>ACM</i> , 131–136. doi:1-59593-087-6/05/0007

(table continues)

No.	Reference
27	Inoue, S., & Yamada, S. (2011). Software reliability growth modeling frameworks with change of testing-environment. <i>International Journal of Reliability, Quality & Safety Engineering</i> , 18(4), 365-376. Retrieved from Computers & Applied Sciences Complete database.
28	Kadry, S. (2011). A new proposed technique to improve software regression testing cost. <i>International Journal of Security & Its Applications</i> , 5(3), 45-58. Retrieved from Computers & Applied Sciences Complete database.
29	Kadry, S., & Kalakech, A. (2011). Cost-effectiveness of regression testing: problem and solution. <i>International Review on Computers & Software</i> , 6(3), 328–335. Retrieved from Computers & Applied Sciences Complete database.
30	Kim, J., Sung, D., & Hong, J. (2011). Performance testing of composite web-service with aspect-based WS-BPEL extension. <i>KSII Transactions on Internet & Information Systems</i> , 5(10), 1841-1860. doi:10.3837/tiis.2011.10.010
31	Klaib, M., Muthuraman, S., Ahmad, N., & Sidek, R. (2010). Tree based test case generation and cost calculation strategy for uniform parametric pairwise testing. <i>Journal of Computer Science</i> , 6(5), 542–547. Retrieved from Computers & Applied Sciences Complete database.
32	Kuhn, R., Wallace, D., & Gallo, A. (2004). Software fault interactions and implications for software testing. <i>IEEE IT Professional</i> , 30(6), 418–421. Retrieved from http://www.nist.gov/index.html .
33	Kuhn, R., Kacker, R., Lei, Y., & Hunter, J. (2009). Combinatorial software testing. <i>IEEE Computer Society</i> , 42(8), 94–96. Retrieved from http://www.nist.gov/index.html .
34	Kuhn, R., Kacker, R., & Lei, Y. (2008). Automated combinatorial test methods: Beyond pairwise testing. <i>Crosstalk Journal of Defense Software Engineering</i> , 21(6), 94–96. Retrieved from http://www.nist.gov/index.html .
35	Kuhn, R., Lei, Y., & Kacker, R. (2008). Practical combinatorial testing: Beyond pairwise. <i>IEEE Computer Society</i> , 10(3), 19–23. Retrieved from http://www.nist.gov/index.html .
36	Kuhn, R., Kacker, R., & Lei, Y. (2009). <i>Random vs combinatorial methods for discrete event simulation of a grid computer network</i> . National Institute of Standard and Technology. Retrieved from http://www.nist.gov/index.html .

(table continues)

No.	Reference
37	Kuhn, D.R., & Reilly, M.J. (2002). <i>An investigation of the applicability of design of experiments to software testing proceeding</i> . 27th NASA/IEEE Software Engineering Workshop. Retrieved from http://www.nist.gov/index.html .
38	Lazić, L. (2010). Software testing optimization by advanced quantitative defect management. <i>Computer Science & Information Systems</i> , 7(3), 459-487. doi:10.2298/CSIS090923008L
39	Lazić, L., & Velašević, D. D. (2004). Applying simulation and design of experiments to the embedded software testing process. <i>Journal of Software Testing: Verification & Reliability</i> , 14(4), 257–282. doi:10.1002/stvr.299
40	Lei, Y., Carver, R., & Kung, D. (2007). A combinatorial strategy for testing concurrent programs. <i>Journal of Software Testing, Verification, and Reliability</i> , 17(4), 207–225. Retrieved from Computers & Applied Sciences Complete database.
41	Lun, L., Chi, X., & Ding, X. (2012). Edge coverage analysis for software architecture testing. <i>Journal of Software (1796217X)</i> , 7(5), 1121–1128. doi:10.4304/jsw.7.5.1121-1128
42	Mala, D., Mohan, V. V., & Kamalpriya, M. M. (2010). Automated software test optimisation framework - an artificial bee colony optimisation-based approach. <i>IET Software</i> , 4(5), 334–348. doi:10.1049/iet-sen.2009.0079
43	Montanez, C., Kuhn, D.R., Brady, M., Rivello, R., Reyes, J., & Powers, M.K., (2011). An application of combinatorial methods to conformance testing for document object model events. NIST-IR 7773. 25 Jan 2011. Retrieved from http://csrs.nist.gov/groups/SNS/acts/documents/DOM-IR-nnnn-110125.doc .
44	Mouchawrab, S., Briand, L. C., Labiche, Y., & Di Penta, M. (2011). Assessing, comparing, and combining state machine-based testing and structural testing: A series of experiments. <i>IEEE Transactions On Software Engineering</i> , 37(2), 161-187. doi:10.1109/TSE.2010.32
45	Nirpal, P. B., & Kale, K. V. (2012). Genetic algorithm based software testing specifically structural testing for software reliability enhancement. <i>International Journal of Computational Intelligence Techniques</i> , 3(1), 60–64. Retrieved from Computers & Applied Sciences Complete database.

(table continues)

No.	Reference
46	Parsa, S., & Khalilian, A. (2010). On the optimization approach towards test suite minimization. <i>International Journal of Software Engineering & Its Applications</i> , 4(1), 15–28. Retrieved from Computers & Applied Sciences Complete database.
47	Sagarna, R., & Lozano, J. (2005). On the performance of estimation of distribution algorithms applied to software testing. <i>Applied Artificial Intelligence</i> , 19(5), 457-489. doi:10.1080/08839510590917861
48	Zheng, Q., & Dan-ping, W. (2009). Software testing effectiveness modeling-based on complete or missing detection data. <i>Journal of Communication & Computer</i> , 6(4), 14–19. Retrieved from Computers & Applied Sciences Complete database.

Appendix E: Cumulative Statistics for Studies where DOE was applied.

Table E1

Data Characteristics Raw Data for Studies That Applied DOE Techniques

Studies	Findings	Sample Size	Reporting Measure	DOE Techniques
Ahmed, B. S., & Zamli, K. Z. (2011)	70	48	Defects	Covering arrays
Alshraideh, M., Mahafzah, B. A., & Al-Sharaeh, S. (2011)	40% more	48	Defects/hour	Factor covering Combination DOE strategy
Baier, C., Haverkort, B. R., Hermanns, H., & Katoen, J. (2010)	25	48	Defects	
Baluda, M., Braione, P., Denaro, G., & Pezzè, M. (2011)	8	48	Defects/hour	Covering arrays
Bandurek, G. R. (2005)	49	48	Defects	Factor covering
Belli, F., Budnik, C., & White, L. (2006)	24% less	48	Test Time Phase	Taguchi Approach
Berling, T., & Runeson, P. (2003)	45	48	Detection	Fractional Factorial design
Bida, A. S. (2009)	50%	48	Defects	Taguchi Approach
Bryce, R., & Colbourn, C. J. (2006)	95%	48	Test Time	Factor coverage
Bryce, R., & Colbourn, C. J. (2007)	90% less	48	Test Time	Pairwise interaction
Bryce, R. C., & Colbourn, C. J. (2009)	26.9 % less	48	Test Time	Covering arrays
Cai, K., Zhao, D., Liu, K., & Bai, C. (2007)	45 % less	48	Test Time	Taguchi Approach
Cangussu, J. W., Cooper, K., & Wong, W. (2009)	47	48	Defects	Taguchi Approach
Cohen, D., Dalal, S., Fredman, M., & Patton, G. (1997)	20 % less	48	Test Time	Combinatorial

(table continues)

Studies	Findings	Sample Size	Reporting Measure	DOE Techniques
Cohen, M., Dwyer, M., & Shi, J. (2008)	20 % less	48	Test Time Phase	Combinatorial
Dalal, S.R., & Mallows, C.L. (1998)	43	48	Detection	Factor covering
Devaraj, E. E., Kumar, S. S., Kavi, T. T., & Rajani Kanth, K. K. (2011)	40% less	48	Test Time	Factor covering
Hartman, A., & Raskin, L. (2003)	5	48	Defects/hour	Covering arrays
He, Z., Staples, G., Ross, M., Court, I., & Hazzard, K. (1997)	33% less	48	Test Time	Taguchi Approach
Hoskins, D.S, Colburn, C.J., & Montgomery, D.C. (2005)	38%	48	Test Time	Covering arrays
Inoue, S., & Yamada, S. (2011)	34	48	Defects Phase	Taguchi Approach
Kadry, S. (2011)	50	48	Detection	Covering arrays
Kadry, S., & Kalakech, A. (2011)	50	48	Defects/hour	Covering arrays
Kim, J., Sung, D. & Hong, J. (2011)	10	48	Defects/hour	Taguchi Approach
Klaib, M., Muthuraman, S., Ahmad, N., & Sidek, R. (2010)	12% less	48	Test Time	Pairwise testing
Kuhn, R., Wallace, D., & Gallo, A. (2004)	30% less	48	Test Time	Combinatorial
Kuhn, R., Kacker, R., Lei, Y., & Hunter, J. (2009)	923	48	Defects Phase	Pairwise interaction
Kuhn, R., Kacker, R., & Lei, Y. (2008)	100	48	Detection Phase	Combinatorial
Kuhn, R., Lei, Y., & Kacker, R. (2008)	93	48	Detection Phase	Combinatorial
Kuhn, R., Kacker, R., & Lei, Y. (2009)	90	48	Detection	2-way combinations

(table continues)

Studies	Findings	Sample Size	Reporting Measure	DOE Techniques
Nirpal, P. B., & Kale, K. V. (2012)	50% less	48	Test Time	Taguchi
Parsa, S., & Khalilian, A. (2010)	27% less	48	Test Time	Taguchi
Sagarna, R., & Lozano, J. (2005)	15% less	48	Test Time	Taguchi
Zheng Q., & Dan-ping, W. (2009)	34% less	48	Test Time	Factor covering

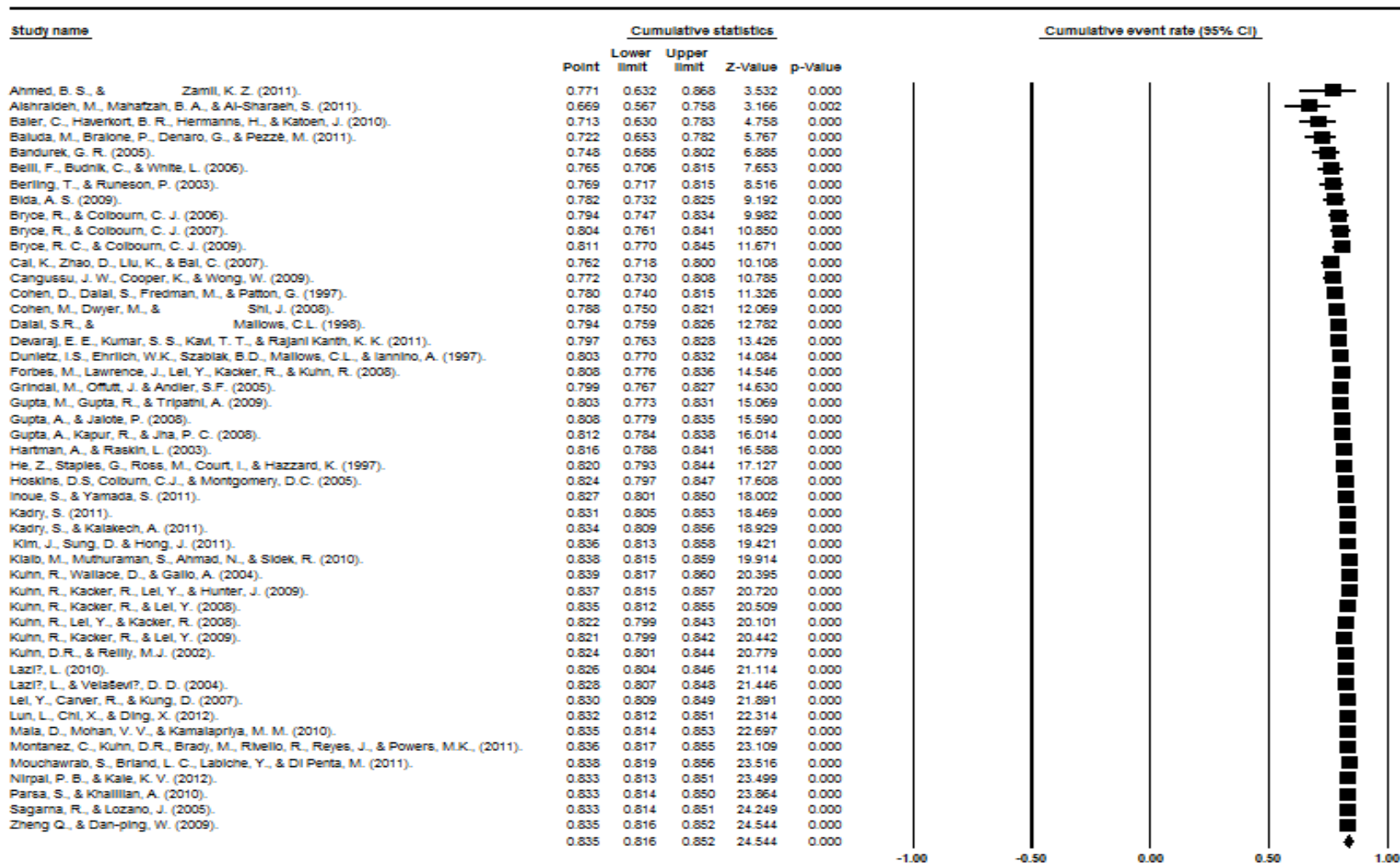


Figure E1. Forest plot of original included studies that had DOE techniques applied.

Appendix F: Cumulative Statistics for Studies where DOE was not applied.

Table F1

Data Characteristics for Studies That Did Not Apply DOE Techniques

Studies	Findings	Sample Size	Reporting Measure
Ahmad, N. N., Khan, M. M., & Rafi, L. S. (2010)	5% more	48	Defects
Alalfi, M. H., Cordy, J. R., & Dean, T. R. (2009)	24	48	Defects
Alsmadi, I. (2012)	84% more	48	Defects
Andrews, J. H., Menzies, T., & Li, F. H. (2011)	90% more	48	Defects
Askarunisa, A., Prameela, P., & Ramraj, N. (2009)	15	48	Defects
Baharom, S., & Shukur, Z. (2008)	20	48	Phase Detection (requirements)
Briand, L.C., Labiche, Y., & He, S. (2009)	20% less	48	Test Time
Bryce, R. C., Sampath, S., & Memon, A. M. (2011)	95% more	48	Defects
Chen, T. T., Lau, M. M., Sim, K. K., & Sun, C. C. (2009)	93% more	48	Defects
Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011)	29	48	Defects
Ciupa, I. I., Pretschner, A. A., Oriol, M. M., Leitner, A. A., & Meyer, B. B. (2011)	66% less	48	Test Time Phase Detection
Clarke, P. J., Power, J. F., Babich, D., & King, T. M. (2012).	55% more	48	(unit/integration)
Foster, G. (2005)	99% less	48	Test Time
Fraser, G., & Arcuri, A. (2013)	62% less	48	Test Time
Goel, A. A., Gupta, S. C., & Wasan, S. K. (2008)	100%	48	Phase Detection (integration)

(table continues)

Studies	Findings	Sample Size	Reporting Measure
Goel, N., & Gupta, M. (2012)	50% more	48	Phase Detection (unit/integration)
Halfond, W. J., Choudhary, S., & Orso, A. (2011)	10 hr less	48	Test Time
Hu, H., Jiang, C., & Cai, K. (2009)	36	48	Phase Detection (regression)
Itkonen, J., Mäntylä, M. V., & Lassenius, C. (2013)	18% less	48	Test Time
Just, R., & Schweiggert, F. (2011)	95 % more	48	Defects
Poon, P., Tse, T. T., Tang, S., & Kuo, F. (2011)	40% less	48	Test Time
Poulding, S., & Clark, J. A. (2010)	5% more	48	Defects
Prakash, V. V., SenthilAnand, N. N., & Bhavani, R. R. (2012)	14% less	48	Test Time
Rao, K., & Sastri, A. (2011)	40% more	48	Defects
Robinson, B., & White, L. (2012)	29% more	48	Defects
Shahbazi, A., Tappenden, A. F., & Miller, J. (2013)	100% more	48	Defects/hour
Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012)	90% more	48	Defects/hour
Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012a)	62% less	48	Test Time
Teasley, B. E., Leventhal, L., Mynatt, C. R., & Rohlman, D. S. (1994)	90% more	48	Defects
Watkins, M. L. (1982)	45% more	48	Defects
Yang, L., Dang, Z., & Fischer, T. (2011)	50% more	48	Defects
Ye, D. (2011)	20% less	48	Test Time
Yoon, H., & Choi, B. (2011)	41% less	48	Test Time
Zhang, Z., & Zhou, Y. (2007)	30% less	48	Test Time
Zielinska, A. (2012)	94% more	48	Defects

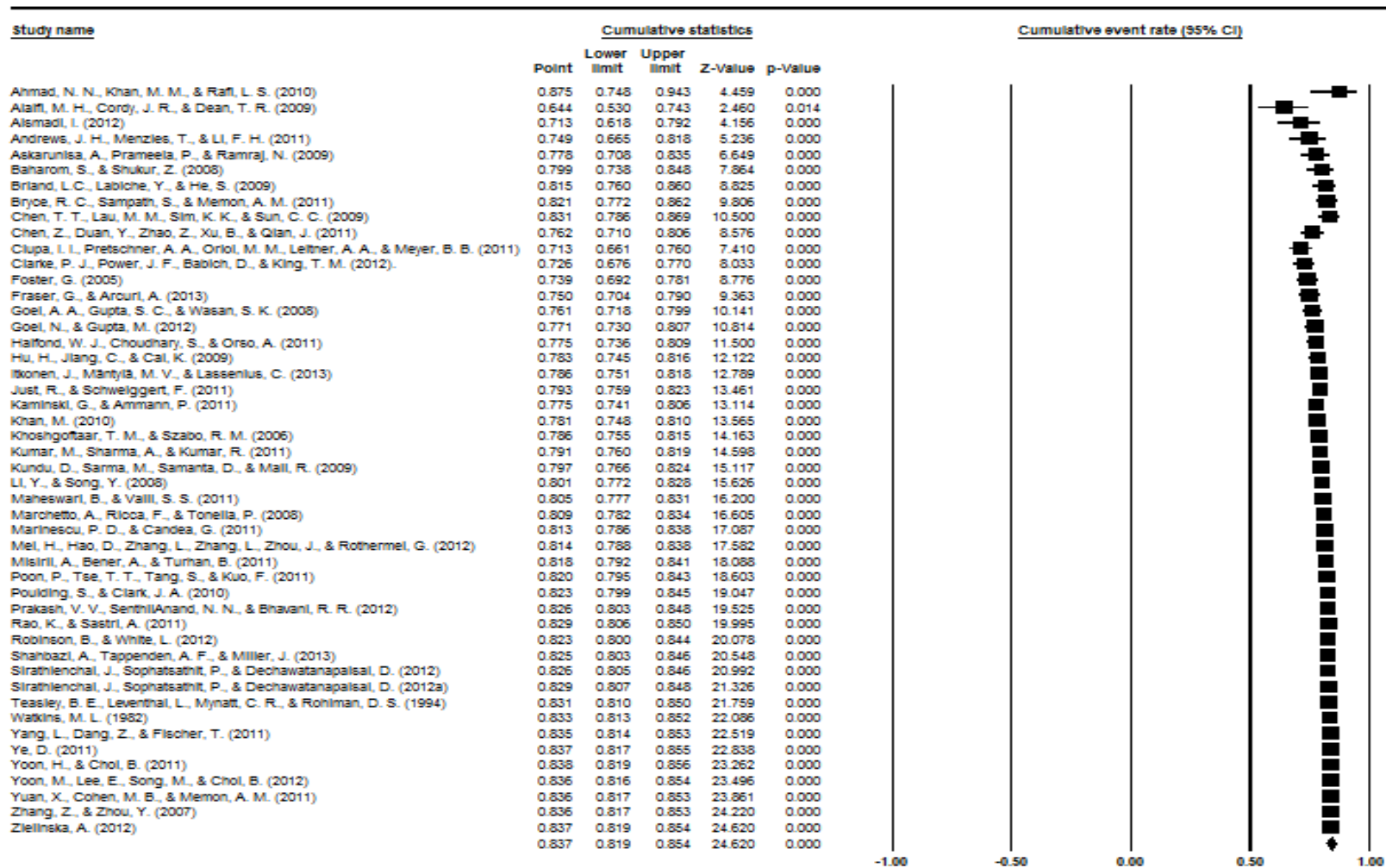


Figure F1. Forest plot of original included studies that did not have DOE techniques applied.

Appendix G: Formulas Used in the CMA Package Computations

The formulas used in the CMA software package for the calculations include the following:

$$\text{Odds ratio} = \frac{\text{TreatedEvnts} * \text{Controlled NonEvents}}{\text{Treated NonEvents} * \text{ControlledEvents}}$$

$$\text{LogOddsRatio} = \ln(\text{OddsRatio})$$

$$d = \text{StdDiff} = \sqrt{3} * \frac{\text{LogOddsRatio}}{\text{Pi}}$$

$$\text{StdDiffSE} = \sqrt{\frac{3 * (\text{LogOddsSE})^2}{(\text{Pi})^2}}$$

$$\text{StdDiffVar} = (\text{StdDiffSE})^2$$

To compute mean differences (g), the program computes a correction factor J,

$$J = 1 - \left(\frac{3}{(4 * df - 1)} \right)$$

Borenstein et al. (2009) g is

$$g = d * J$$

$$\text{stdErr}(g) = \text{StdErr}(d) * J$$

$$\text{Variance}(g) = (\text{StdErr}(g))^2$$

To convert standardized mean difference to correlation, r,

$$r = \frac{d}{\sqrt{d^2 + a}}$$

where a is the correction factor for instances where $n_1 \neq n_2$,

$$a = \frac{(n_1 + n_2)^2}{n_1 n_2}$$

For the Z distribution,

$$Z = \frac{\text{Effect Size}}{\text{Standard Error}}$$

To compute a 95% confidence interval,

Lower Limit = Effect Size - 1.96 (Standard Error)

Upper Limit = Effect Size + 1.96 (Standard Error)

Appendix H: Raw Data

Table H1

Studies Without Design of Experiments Raw Data Calculations

No.	Reference	\bar{X}_1	\bar{X}_2	S ₁	S ₂	n ₁	n ₂	d
1	Ahmad, N. N., Khan, M. M., & Rafi, L. S. (2010)	0.027	0.048	0.215	0.0208	2	2	-0.972
2	Alalfi, M. H., Cordy, J. R., & Dean, T. R. (2009)	0.500	0.400	0.530	0.520	8	8	0.1905
3	Alsmadi, I. (2012)	0.785	0.798	0.070	0.050	4	4	-0.2138
4	Andrews, J. H., Menzies, T., & Li, F. H. (2011)	0.844	0.842	0.260	0.257	16	16	0.0077
5	Askarunisa, A., Prameela, P., & Ramraj, N. (2009)	29.500	12.500	7.600	3.000	3	3	4.9658
6	Baharom, S., & Shukur, Z. (2008)	2.750	4.000	1.300	3.200	4	4	-0.511
7	Briand, L.C., Labiche, Y., & He, S. (2009)	36.250	35.750	28.700	27.900	4	4	0.0177
8	Bryce, R. C., Sampath, S., & Memon, A. M. (2011)	84.000	95.000	14.300	2.600	13	13	-1.0703
9	Chen, T. T., Lau, M. M., Sim, K. K., & Sun, C. C. (2009)	402.00 0	385.000	213.400	206.100	10	10	0.0811
10	Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011)	0.250	0.210	0.260	0.270	30	30	0.1509
11	Ciupa, I. I., Pretschner, A. A., Oriol, M. M., Leitner, A. A., & Meyer, B. B. (2011)	14.000	82.500	5.600	28.900	2	2	-3.2908

(table continues)

No.	Reference	\bar{X}_1	\bar{X}_2	S_1	S_2	n_1	n_2	d
12	Clarke, P. J., Power, J. F., Babich, D., & King, T. M. (2012).	208.75 0	0.708	300.700	0.160	16	16	14.7761
13	Foster, G. (2005)	6.310	8.770	2.990	0.600	13	13	-1.1194
14	Fraser, G., & Arcuri, A. (2013)	37.650	33.400	42.260	36.980	20	20	0.107
15	Goel, A. A., Gupta, S. C., & Wasan, S. K. (2008)	3.540	6.940	1.400	1.300	4	4	-1.497
16	Goel, N., & Gupta, M. (2012)	1.000	2.000	1.000	1.000	3	3	-0.5
17	Halfond, W. J., Choudhary, S., & Orso, A. (2011)	24.330	989.200	25.340	696.190	9	9	-0.6354
18	Hu, H., Jiang, C., & Cai, K. (2009)	-5.600	-16.400	11.600	10.200	5	5	0.9888
19	Itkonen, J., Mäntylä, M. V., & Lassenius, C. (2013)	13.600	16.000	7.100	16.700	5	8	-0.187
20	Just, R., & Schweiggert, F. (2011)	86.710	56.990	3.900	15.300	5	5	2.662
21	Kaminski, G., & Ammann, P. (2011)	82.200	99.100	18.800	1.600	5	5	-1.2667
22	Khan, M. (2010)	0.500	1.000	0.600	0.000	5	5	-1.1784
23	Khoshgoftaar, T. M., & Szabo, R. M. (2006)	1.200	0.590	3.200	0.800	40	40	0.0002
24	Kumar, M., Sharma, A., & Kumar, R. (2011)	1.000	1.000	0.000	0.000	6	9	0
25	Kundu, D., Sarma, M., Samanta, D., & Mall, R. (2009)	58.700	59.900	28.200	28.500	7	7	-0.0423
26	Li, Y., & Song, Y. (2008)	50.500	94.130	36.930	58.340	8	8	-0.8936
27	Maheswari, B., & Valli, S. S. (2011)	50.000	75.000	53.500	26.700	8	8	-0.5913
28	Marchetto, A., Ricca, F., & Tonella, P. (2008)	30.500	55.300	6.140	17.370	4	4	-1.9037
29	Marinescu, P. D., & Candea, G. (2011)	70.000	71.000	12.400	12.600	2	2	-0.08

(table continues)

No.	Reference	\bar{X}_1	\bar{X}_2	S_1	S_2	n_1	n_2	d
30	Mei, H., Hao, D., Zhang, L., Zhang, L., Zhou, J., & Rothermel, G. (2012)	-4.120	-12.300	3.290	28.990	4	4	-3.3048
31	Misirli, A., Bener, A., & Turhan, B. (2011)	0.818	0.742	0.050	0.180	5	5	0.5753
32	Poon, P., Tse, T. T., Tang, S., & Kuo, F. (2011)	1.430	0.800	1.550	0.350	5	5	0.4895
33	Poulding, S., & Clark, J. A. (2010)	0.730	0.760	0.190	0.240	5	5	-0.1389
34	Prakash, V. V., SenthilAnand, N. N., & Bhavani, R. R. (2012)	1.300	2.700	0.580	0.580	5	5	-1.9084
35	Rao, K., & Sastri, A. (2011)	0.250	1.000	0.260	0.000	14	14	-4.0805
36	Robinson, B., & White, L. (2012)	1.000	0.700	0.630	0.820	6	6	0.4103
37	Shahbazi, A., Tappenden, A. F., & Miller, J. (2013)	1.014	1.014	0.00044	0.00041	4	4	0
38	Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012)	79.088	78.397	44.966	24.921	9	9	3.918
39	Sirathienchai, J., Sophatsathit, P., & Dechawatanapaisal, D. (2012)	49.900	53.300	21.700	7.700	9	9	-0.2088
40	Teasley, B. E., Leventhal, L., Mynatt, C. R., & Rohlman, D. S. (1994)	5.710	5.300	2.600	3.200	17	10	0.1413
41	Watkins, M. L. (1982)	3831.400	-3831.800	2864.980	2872.260	10	10	0.0001
42	Yang, L., Dang, Z., & Fischer, T. (2011)	0.630	0.670	0.140	0.190	6	6	-0.2395
43	Ye, D. (2011)	2.500	4.500	2.100	2.100	2	2	-0.9524
44	Yoon, H., & Choi, B. (2011)	153.500	159.500	170.700	95.200	10	10	-0.0302
45	Yoon, M., Lee, E., Song, M., & Choi, B. (2012)	70.300	94.900	21.800	4.100	8	8	-1.5684

(table continues)

No.	Reference	\bar{X}_1	\bar{X}_2	S ₁	S ₂	n ₁	n ₂	d
46	Yuan, X., Cohen, M. B., & Memon, A. M. (2011)	0.770	0.920	0.740	0.690	6	6	-0.2097
47	Zhang, Z., & Zhou, Y. (2007)	0.880	0.750	0.200	0.280	12	12	0.5343
48	Zielińska, A. (2012)	40.000	140.000	28.300	56.600	2	2	-2.2348

Table H2

Studies With Design of Experiments Raw Data Calculations

No.	Reference	\bar{X}_1	\bar{X}_2	S_1	S_2	n_1	n_2	d
1	Ahmed, B. S., & Zamli, K. Z. (2011)	0.025	5.000	0.700	0.000	2	2	0.2470
2	Alshraideh, M., Mahafzah, B. A., & Al-Sharaeh, S. (2011)	347.500	155.600	158.400	69.700	8	8	1.5740
3	Baier, C., Haverkort, B. R., Hermanns, H., & Katoen, J. (2010)	299.000	457.600	401.000	520.500	4	5	1.0847
4	Baluda, M., Braione, P., Denaro, G., & Pezzè, M. (2011)	36.600	107.200	22.200	6.500	20	20	-1.0320
5	Bandurek, G. R. (2005)	15.800	4.000	7.500	0.000	5	5	2.2250
6	Belli, F., Budnik, C., & White, L. (2006)	26.000	30.000	6.600	5.600	3	3	0.6570
7	Berling, T., & Runeson, P. (2003)	9.000	10.000	10.600	12.500	16	16	-0.1130
8	Bida, A. S. (2009)	1.000	0.600	0.000	0.550	5	5	1.0285
9	Bryce, R., & Colbourn, C. J. (2006)	0.900	0.750	0.110	0.010	3	3	1.9200
10	Bryce, R., & Colbourn, C. J. (2007)	88.600	79.800	1.610	1.170	5	5	7.0230
11	Bryce, R. C., & Colbourn, C. J. (2009)	159.800	160.000	43.900	43.200	6	6	0.0045
12	Cai, K., Zhao, D., Liu, K., & Bai, C. (2007)	37.500	3.300	21.800	2.400	4	4	2.2053
13	Cangussu, J. W., Cooper, K., & Wong, W. (2009)	13.600	11.600	8.080	7.230	3	3	0.2608
14	Cohen, D., Dalal, S., Fredman, M., & Patton, G. (1997)	54.000	75.000	13.000	19.000	4	4	1.2900
15	Cohen, M., Dwyer, M., & Shi, J. (2008)	149.000	169.000	71.900	74.900	5	5	0.2791

(table continues)

No.	Reference	\bar{X}_1	\bar{X}_2	S_1	S_2	n_1	n_2	d
16	Dalal, S.R., & Mallows, C.L. (1998)	0.905	0.686	0.940	0.180	27	27	0.4803
17	Devaraj, E. E., Kumar, S. S., Kavi, T. T., & Rajani Kanth, K. K. (2011)	4.490	1.234	0.990	1.910	8	8	4.6300
18	Dunietz, I.S., Ehrlich, W.K., Szablak, B.D., Mallows, C.L., & Iannino, A. (1997)	12.170	9.000	7.500	9.900	6	6	0.3610
19	Forbes, M., Lawrence, J., Lei, Y., Kacker, R., & Kuhn, R. (2008)	256.000	217.000	130.800	101.700	7	7	0.3329
20	Grindal, M., Offutt, J. & Andler, S.F. (2005)	58.200	37.700	18.910	13.040	4	4	1.2623
21	Gupta, M., Gupta, R., & Tripathi, A. (2009)	144.800	115.800	269.000	208.000	5	5	0.1208
22	Gupta, A., & Jalote, P. (2008)	0.710	2.070	0.182	0.605	5	5	-3.0430
23	Gupta, A., Kapur, R., & Jha, P. C. (2008)	54.310	15.400	11.420	1.750	2	16	7.1900
24	Hartman, A., & Raskin, L. (2003)	54.600	43.400	71.180	45.920	3	3	0.1870
25	He, Z., Staples, G., Ross, M., Court, I., & Hazzard, K. (1997)	1.000	1.000	1.000	0.000	3	3	0.0000
26	Hoskins, D.S, Colburn, C.J., & Montgomery, D.C. (2005).	58.300	41.600	21.460	7.960	66	3	1.0370
27	Inoue, S., & Yamada, S. (2011)	19.200	16.900	11.300	11.900	6	6	0.1983
28	Kadry, S. (2011)	24.500	27.000	14.850	32.530	2	7	-0.0988
29	Kadry, S., & Kalakech, A. (2011)	32.500	27.000	38.890	32.530	2	2	0.1534
30	Kim, J., Sung, D. & Hong, J. (2011)	2169.230	1502.390	399.500	358.200	7	7	2.4840
31	Klaib, M., Muthuraman, S., Ahmad, N., & Sidek, R. (2010)	45.000	19.000	50.900	18.400	2	2	0.7002

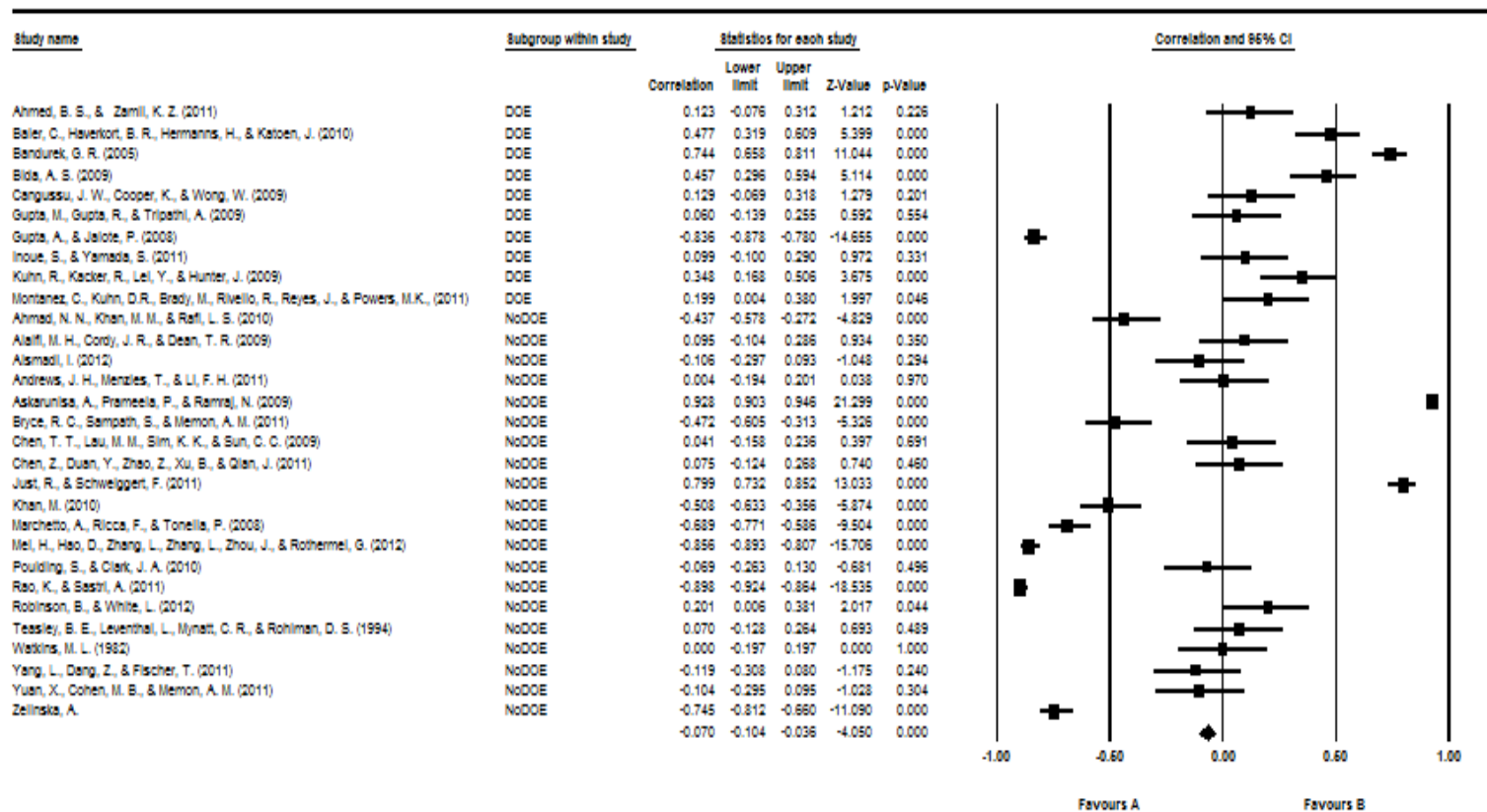
(table continues)

No.	Reference	\bar{X}_1	\bar{X}_2	S_1	S_2	n_1	n_2	d
32	Kuhn, R., Wallace, D., & Gallo, A. (2004)	82.170	89.750	22.400	14.800	6	4	-0.3811
33	Kuhn, R., Kacker, R., Lei, Y., & Hunter, J. (2009)	93.170	80.330	13.900	20.100	6	6	0.7431
34	Kuhn, R., Kacker, R., & Lei, Y. (2008)	1.000	1.000	0.900	0.900	9	9	0.0000
35	Kuhn, R., Lei, Y., & Kacker, R. (2008)	93.200	82.500	13.110	22.250	6	6	0.5847
36	Kuhn, R., Kacker, R., & Lei, Y. (2009)	5.300	4.600	7.600	4.900	3	3	0.1094
37	Kuhn, D.R., & Reilly, M.J. (2002)	16.700	16.700	18.800	16.200	6	6	0.0000
38	Lazic, L. (2010)	15.600	12.200	7.500	9.200	6	8	295.4000
39	Lazic, L., & Velašević, D. D. (2004)	7.700	6.600	3.500	3.100	9	9	0.3330
40	Lei, Y., Carver, R., & Kung, D. (2007)	1093	235084	2293	523785	5	5	-0.7060
41	Lun, L., Chi, X., & Ding, X. (2012)	47.400	55.100	0.800	12.400	5	5	-0.8750
42	Mala, D., Mohan, V. V., & Kamalpriya, M. M. (2010)	181.500	127.500	144.000	85.000	10	10	0.4576
43	Montanez, C., Kuhn, D.R., Brady, M., Rivello, R., Reyes, J., & Powers, M.K., (2011)	77.000	61.800	31.700	42.400	6	6	0.4064
44	Mouchawrab, S., Briand, L. C., Labiche, Y., & Di Penta, M. (2011)	67.000	77.000	11.400	8.600	9	9	0.9903
45	Nirpal, P. B., & Kale, K. V. (2012)	1000.000	550.000	0.000	158.000	10	10	12.1600
46	Parsa, S., & Khalilian, A. (2010)	76.100	75.700	13.300	13.400	6	6	0.1199
47	Sagarna, R., & Lozano, J. (2005)	19.600	17.400	3.300	2.100	7	7	0.7857
48	Zheng Q., & Dan-ping, W. (2009)	0.500	0.900	0.150	0.030	5	5	0.0571

Appendix I: Effectiveness Measures Statistical Data

Model	Group by	Study name	Subgroup	Statistics for each study					Correlation and 95% CI				
				Correlation	Lower limit	Upper limit	Z-Value	p-Value	-0.250	-0.125	0.000	0.125	0.250
	DOE	Ahmed, B.	DOE	0.123	-0.076	0.312	1.212	0.226					
	DOE	Baier, C.,	DOE	0.477	0.319	0.609	5.399	0.000					
	DOE	Bandurek,	DOE	0.744	0.658	0.811	11.044	0.000					
	DOE	Bida, A. S.	DOE	0.457	0.296	0.594	5.114	0.000					
	DOE	Cangussu, J.	DOE	0.129	-0.069	0.318	1.279	0.201					
	DOE	Gupta, M.,	DOE	0.060	-0.139	0.255	0.592	0.554					
	DOE	Gupta, A.,	DOE	-0.836	-0.878	-0.780	-14.655	0.000					
	DOE	Inoue, S., &	DOE	0.099	-0.100	0.290	0.972	0.331					
	DOE	Kuhn, R.,	DOE	0.348	0.168	0.506	3.675	0.000					
	DOE	Montanez,	DOE	0.199	0.004	0.380	1.997	0.046					
Fixed	DOE			0.142	0.084	0.200	4.717	0.000					
	NoDOE	Ahmad, N.	NoDOE	-0.437	-0.578	-0.272	-4.829	0.000					
	NoDOE	Alalfi, M.	NoDOE	0.095	-0.104	0.286	0.934	0.350					
	NoDOE	Alsmadi, I.	NoDOE	-0.106	-0.297	0.093	-1.048	0.294					
	NoDOE	Andrews, J.	NoDOE	0.004	-0.194	0.201	0.038	0.970					
	NoDOE	Askarunisa,	NoDOE	0.928	0.903	0.946	21.299	0.000					
	NoDOE	Bryce, R.	NoDOE	-0.472	-0.605	-0.313	-5.326	0.000					
	NoDOE	Chen, T. T.,	NoDOE	0.041	-0.158	0.236	0.397	0.691					
	NoDOE	Chen, Z.,	NoDOE	0.075	-0.124	0.268	0.740	0.460					
	NoDOE	Just, R., &	NoDOE	0.799	0.732	0.852	13.033	0.000					
	NoDOE	Khan, M.	NoDOE	-0.508	-0.633	-0.356	-5.874	0.000					
	NoDOE	Marchetto,	NoDOE	-0.689	-0.771	-0.586	-9.504	0.000					
	NoDOE	Mei, H.,	NoDOE	-0.856	-0.893	-0.807	-15.706	0.000					
	NoDOE	Poulding, S.,	NoDOE	-0.069	-0.263	0.130	-0.681	0.496					
	NoDOE	Rao, K., &	NoDOE	-0.898	-0.924	-0.864	-18.535	0.000					
	NoDOE	Robinson,	NoDOE	0.201	0.006	0.381	2.017	0.044					
	NoDOE	Teasley, B.	NoDOE	0.070	-0.128	0.264	0.693	0.489					
	NoDOE	Watkins, M.	NoDOE	0.000	-0.197	0.197	0.000	1.000					
	NoDOE	Yang, L.,	NoDOE	-0.119	-0.308	0.080	-1.175	0.240					
	NoDOE	Yuan, X.,	NoDOE	-0.104	-0.295	0.095	-1.028	0.304					
	NoDOE	Zielh?ska, A.	NoDOE	-0.745	-0.812	-0.660	-11.090	0.000					
Fixed	NoDOE			-0.170	-0.210	-0.130	-8.183	0.000					
Fixed	Overall			-0.070	-0.104	-0.036	-4.050	0.000					

Figure II. Computed statistical data for studies reporting findings as detected defects.

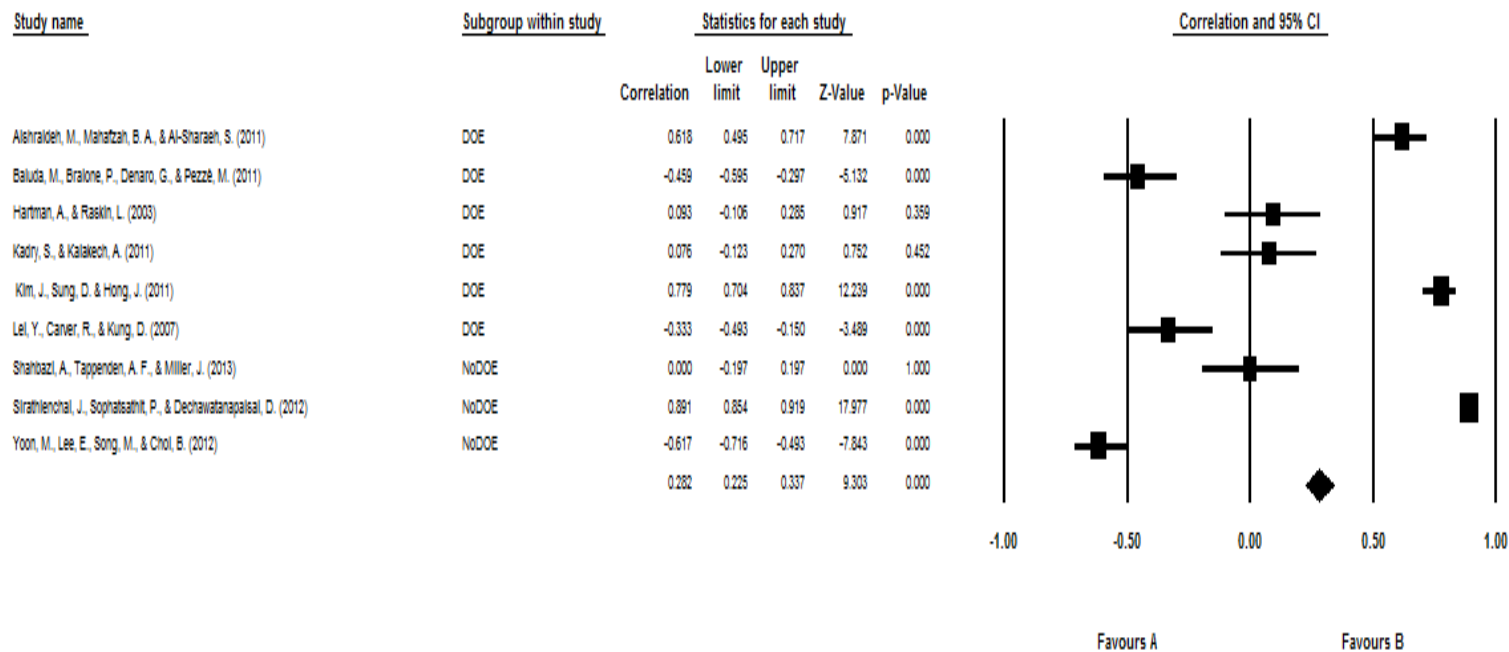


Meta Analysis

Figure 12. Forest plot studies reporting findings as detected defects.
(Note. A is the DOE subgroup and B is the NoDOE subgroup.)

Model	Group by	Study name	Subgroup	Statistics for each study					Correlation and 95% CI					
				Correlation	Lower limit	Upper limit	Z-Value	p-Value	-0.250	-0.125	0.000	0.125	0.250	
	DOE	Alshraideh,	DOE	0.618	0.495	0.717	7.871	0.000						
	DOE	Bahuda, M.,	DOE	-0.459	-0.595	-0.297	-5.132	0.000						
	DOE	Hartman, A.,	DOE	0.093	-0.106	0.285	0.917	0.359						
	DOE	Kadry, S., &	DOE	0.076	-0.123	0.270	0.752	0.452						
	DOE	Kim, J.,	DOE	0.779	0.704	0.837	12.239	0.000						
	DOE	Lei, Y.,	DOE	-0.333	-0.493	-0.150	-3.489	0.000						
Fixed	DOE			0.235	0.161	0.306	6.139	0.000						
	NoDOE	Shahbazi, A.,	NoDOE	0.000	-0.197	0.197	0.000	1.000						
	NoDOE	Sirathienchai,	NoDOE	0.891	0.854	0.919	17.977	0.000						
	NoDOE	Yoon, M.,	NoDOE	-0.617	-0.716	-0.493	-7.843	0.000						
Fixed	NoDOE			0.361	0.270	0.446	7.312	0.000						
Fixed	Overall			0.282	0.225	0.337	9.303	0.000						

Figure 13. Computed statistical data for studies that reported defect detection rate.

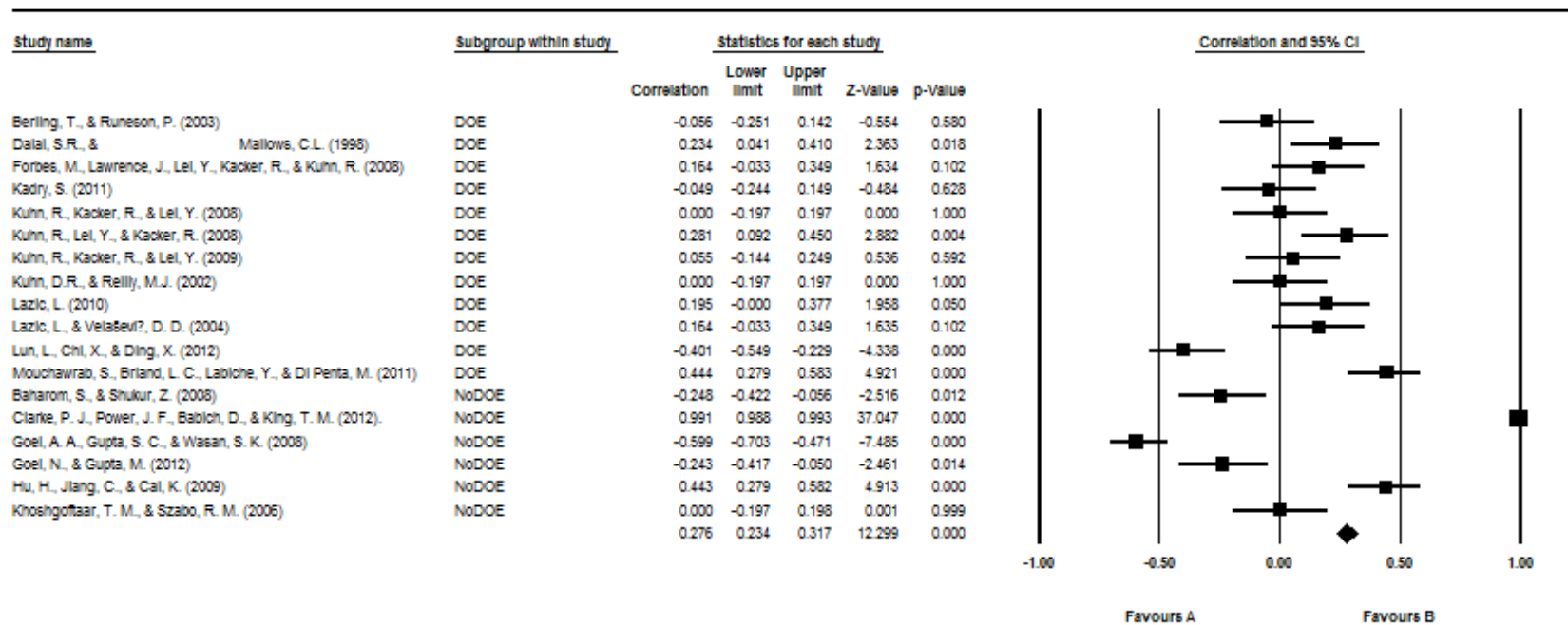


Meta Analysis

Figure 14. Forest plot for studies reporting findings as defect detection rate. (Note. A is the DOE subgroup and B is the NoDOE subgroup.)

Model	Group by Subgroup	Study name	Subgroup within study	Statistics for each study					Correlation and 95% CI					
				Correlation	Lower limit	Upper limit	Z-Value	p-Value	-1.00	-0.50	0.00	0.50	1.00	
	DOE	Berling, T.	DOE	-0.056	-0.251	0.142	-0.554	0.580						
	DOE	Delel, S.R.	DOE	0.234	0.041	0.410	2.363	0.018						
	DOE	Forbes, M.	DOE	0.164	-0.033	0.349	1.634	0.102						
	DOE	Kadry, S.	DOE	-0.049	-0.244	0.149	-0.484	0.628						
	DOE	Kuhn, P.	DOE	0.000	-0.197	0.197	0.000	1.000						
	DOE	Kuhn, R.	DOE	0.281	0.092	0.450	2.882	0.004						
	DOE	Kuhn, R.	DOE	0.055	-0.144	0.249	0.536	0.592						
	DOE	Kuhn, D.R.	DOE	0.000	-0.197	0.197	0.000	1.000						
	DOE	Lazic, L.	DOE	1.000	1.000	1.000	78.818	0.000						
	DOE	Lazic, L. &	DOE	0.164	-0.033	0.349	1.635	0.102						
	DOE	Lun, L. Chi	DOE	-0.401	-0.549	-0.229	-4.338	0.000						
	DOE	Mouchewra	DOE	0.444	0.279	0.583	4.921	0.000						
Fixed	DOE			0.727	0.700	0.752	32.976	0.000						
	NoDOE	Baharom,	NoDOE	-0.248	-0.422	-0.056	-2.516	0.012						
	NoDOE	Clarke, P.	NoDOE	0.991	0.988	0.993	37.047	0.000						
	NoDOE	Goel, A. A.	NoDOE	-0.599	-0.703	-0.471	-7.485	0.000						
	NoDOE	Goel, N. &	NoDOE	-0.243	-0.417	-0.050	-2.461	0.014						
	NoDOE	Hu, H.	NoDOE	0.443	0.279	0.582	4.913	0.000						
	NoDOE	Khoshgotte	NoDOE	0.000	-0.197	0.198	0.001	0.999						
Fixed	NoDOE			0.542	0.488	0.592	16.146	0.000						
Fixed	Overall			0.670	0.645	0.694	36.096	0.000						

Figure 15. Computed statistical data for studies that reported defects by phase detected.

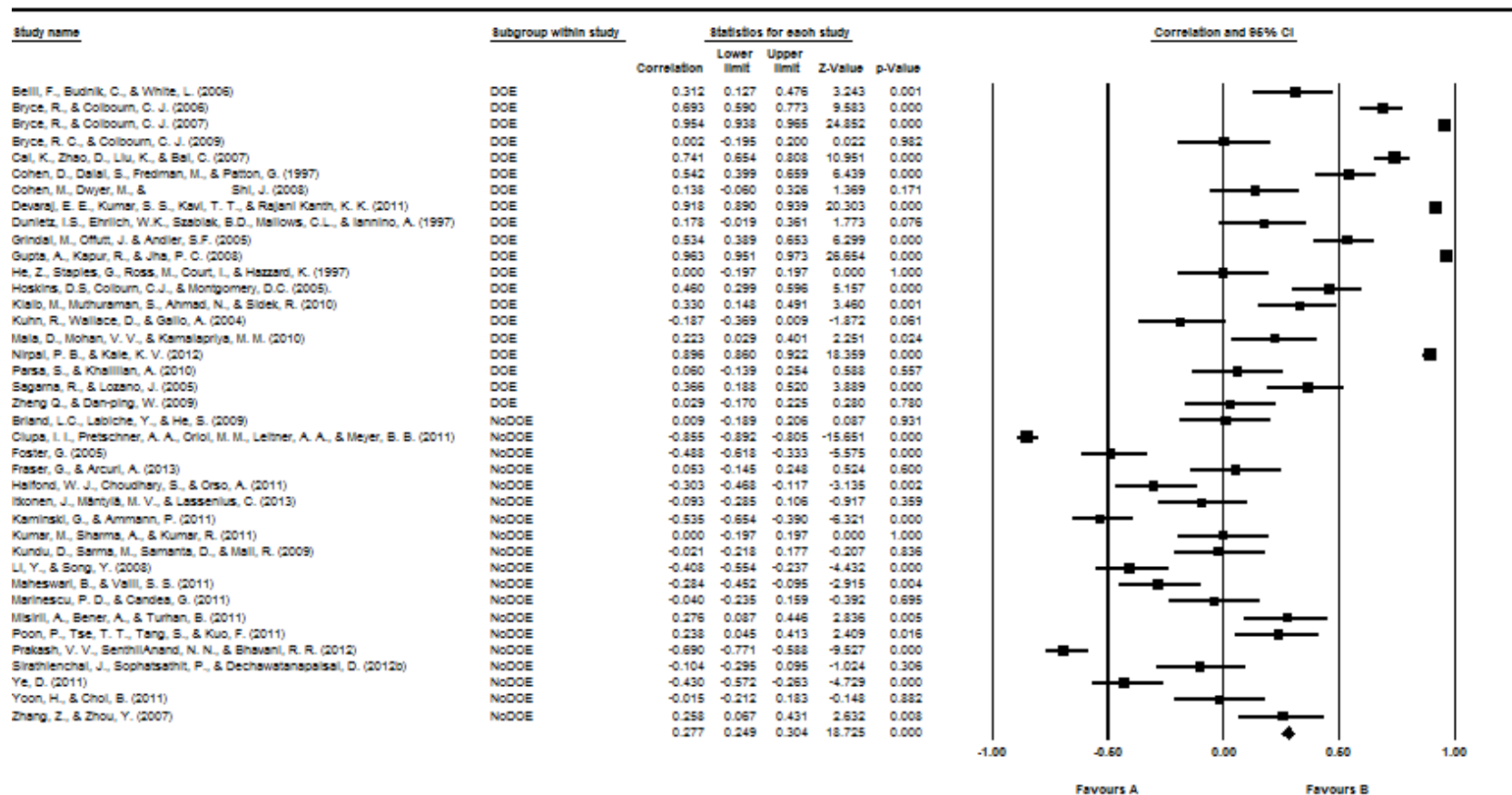


Meta Analysis

Figure 16. Forest plot for studies that reported defects by phase detected.
(Note. A is the DOE subgroup and B is the NoDOE subgroup.)

Model	Group by Subgroup	Study name	Subgroup within study	Statistics for each study					Correlation and 95% CI				
				Correlation	Lower limit	Upper limit	Z-Value	p-Value	-0.250	-0.125	0.000	0.125	0.250
	DOE	Bryce, R. C.,	DOE	0.002	-0.195	0.200	0.022	0.982					
	DOE	Cai, K.,	DOE	0.741	0.654	0.808	10.951	0.000					
	DOE	Cohen, D., D	DOE	0.542	0.399	0.659	6.439	0.000					
	DOE	Cohen, M.,	DOE	0.138	-0.060	0.326	1.369	0.171					
	DOE	Devaraj, E.	DOE	0.918	0.890	0.939	20.303	0.000					
	DOE	Dunietz, I.S.,	DOE	0.178	-0.019	0.361	1.773	0.076					
	DOE	Grindal, M.,	DOE	0.534	0.389	0.653	6.299	0.000					
	DOE	Gupta, A.,	DOE	0.963	0.951	0.973	26.654	0.000					
	DOE	He,	DOE	0.000	-0.197	0.197	0.000	1.000					
	DOE	Hoskins, D.S,	DOE	0.460	0.299	0.596	5.157	0.000					
	DOE	Klaib, M.,	DOE	0.330	0.148	0.491	3.460	0.001					
	DOE	Kuhn, R.,	DOE	-0.187	-0.369	0.009	-1.872	0.061					
	DOE	Mala, D.,	DOE	0.223	0.029	0.401	2.251	0.024					
	DOE	Nirpal, P. B.,	DOE	0.896	0.860	0.922	18.359	0.000					
	DOE	Parsa, S., &	DOE	0.060	-0.139	0.254	0.588	0.557					
	DOE	Sagarna, R.,	DOE	0.366	0.188	0.520	3.889	0.000					
	DOE	Zheng Q., &	DOE	0.029	-0.170	0.225	0.280	0.780					
Fixed	DOE			0.632	0.607	0.656	36.146	0.000					
	NoDOE	Briand, L.C.,	NoDOE	0.009	-0.189	0.206	0.087	0.931					
	NoDOE	Ciupa, I. I.,	NoDOE	-0.855	-0.892	-0.805	-15.651	0.000					
	NoDOE	Foster, G.	NoDOE	-0.488	-0.618	-0.333	-5.575	0.000					
	NoDOE	Fraser, G., &	NoDOE	0.053	-0.145	0.248	0.524	0.600					
	NoDOE	Halfond, W.	NoDOE	-0.303	-0.468	-0.117	-3.135	0.002					
	NoDOE	Itkonen, J.,	NoDOE	-0.093	-0.285	0.106	-0.917	0.359					
	NoDOE	Kaminski,	NoDOE	-0.535	-0.654	-0.390	-6.321	0.000					
	NoDOE	Kumar, M.,	NoDOE	0.000	-0.197	0.197	0.000	1.000					
	NoDOE	Kundu, D.,	NoDOE	-0.021	-0.218	0.177	-0.207	0.836					
	NoDOE	Li, Y., &	NoDOE	-0.408	-0.554	-0.237	-4.432	0.000					
	NoDOE	Maheswari,	NoDOE	-0.284	-0.452	-0.095	-2.915	0.004					
	NoDOE	Marinescu, P.	NoDOE	-0.040	-0.235	0.159	-0.392	0.695					
	NoDOE	Misirli, A.,	NoDOE	0.276	0.087	0.446	2.836	0.005					
	NoDOE	Poon, P.,	NoDOE	0.238	0.045	0.413	2.409	0.016					
	NoDOE	Prakash, V.	NoDOE	-0.690	-0.771	-0.588	-9.527	0.000					
	NoDOE	Sirathienchai	NoDOE	-0.104	-0.295	0.095	-1.024	0.306					
	NoDOE	Ye, D.	NoDOE	-0.430	-0.572	-0.263	-4.729	0.000					
	NoDOE	Yoon, H., &	NoDOE	-0.015	-0.212	0.183	-0.148	0.882					
	NoDOE	Zhang, Z., &	NoDOE	0.258	0.067	0.431	2.632	0.008					
Fixed	NoDOE			-0.258	-0.298	-0.216	-11.719	0.000					
Fixed	Overall			0.277	0.249	0.304	18.725	0.000					

Figure 17. Computed statistical data for studies that reported total testing hours.



Meta Analysis

Figure 18. Forest plot for studies that reported total testing hours. (Note. A is the DOE subgroup and B is the NoDOE subgroup.)

Curriculum Vitae

Educational History

Jackson State University	BS	1975	Mathematics
University of Dallas	MBA	1995	Management Information Systems

Professional Experience

1976 - 2003: Manager and Technical Lead, InterVoiceBrite, Cisco Systems, NEC, and GTE

- Led systems support team in the build, integration, and test of interactive voice response (IVR) systems.
- Led team which supported routers software development and test.
- Developed and test central office telephony software.

Supervised, Raytheon Systems

- Led Software Configuration Management team supporting software development, test, and release for mission-critical software.

Senior Technical Analyst, Motorola Government Electronics Group

- Test data reduction for System Integration & Test
- Developed system requirements and performance specifications for pagers.

Programmer Analyst, Honeywell Large Information Systems Division

- Developed Systems software (COBOL and FORTRAN Compilers)

2003 - 2004: Adjunct Instructor, Westwood College of Technology

- Taught Mathematics courses: Developmental Mathematics, Intermediate Mathematics and Algebra

2004 -2009: Senior Configuration Analyst, L3 Communications Link Simulation and Training

- Provided technical for the release of software and trainers supporting multiple projects crossing multiple aircraft platforms for armed services air pilot simulator trainers and schoolhouse training.

2009 – Present: Senior Software Quality Engineer, Raytheon Company

- Manufacturing Center of Excellence site lead for software maintenance, software release, and continuous improvement processes.

Professional Certifications

Software Quality Engineer

American Society for Quality

Publications

Baker, A., Bittner, T., Makrigeorgis, C., Johnson, G. & Haefner, J. (2010). Teaching prospect theory with the deal or no deal game show. *Teaching Statistics: An International Journal for Teachers*, 32(3), 81–87.

Professional Associations

American Society for Quality (ASQ)

Institute for Operations Research and Management Science (INFORMS)

Society of Women Engineers (SWE)

Contact Information: gloria.b.johnson@raytheon.com