SOFTWARE EFFORT ESTIMATION ACCURACY: A COMPARATIVE STUDY OF ESTIMATIONS BASED ON SOFTWARE SIZING AND DEVELOPMENT METHODS

by

Mark T. Lafferty

A Dissertation Presented in Partial Fulfillment

Of the Requirements for the Degree

Doctor of Philosophy

Capella University

May 2010

Abstract

The number of project failures and those projects completed over cost and over schedule has been a significant issue for software project managers. Among the many reasons for failure, inaccuracy in software estimation—the basis for project bidding, budgeting, planning, and probability estimates—has been identified as a root cause of a high percentage of failures. Poor estimates have not only led projects to exceed budget and go over schedule but also, in many cases, to be terminated entirely. The ability to accurately estimate software development projects changes as newer methodologies replace old ones. Research in this area has been sporadic and there has been little research into the root cause of estimation inaccuracy. The purpose of this study was to determine if there was a difference in software effort-estimation accuracy between the development methodologies of waterfall and incremental methodologies, and the newer agile development methodology. Specifically, the impact of using source line of code (SLOC), function point (FP), or story-point sizing methods were explored across three common development methodologies of waterfall, incremental, and agile. The International Software Benchmarking Standards Group (ISBSG) database was analyzed using Statistical Package for the Social Sciences (SPSS) 15.0 software and examined the relationship between the independent variables (IVs) (estimation parameters) and the dependent variable (DV) estimate accuracy of software projects. Though the hypotheses were not able to be tested, the research question was explored by testing the independent variables through one-way ANOVAs. This resulted in no difference in effort accuracy for the two sizing methods SLOC and FP while there were not enough statistical samples of story-point to determine effect. Similarly, there was no difference in effort accuracy for the two development methods waterfall and agile while there were not enough statistical samples of incremental to determine effect. While limitations existed using the ISBSG database, it is recommended that the database becomes a common tool for future research.

Dedication

To my father and mother Robert and Ruth Lafferty

To my wife Helen for her love and support

Acknowledgments

I wish to thank Dr. Don Gottwald, my mentor and Dissertation Chair, for his excellent support, and holding me to meticulous academic standards for this research. Your mentorship enabled me to move through the dissertation process with a quality result. I would like to thank my committee members, Dr. Rick Livingood and Dr. Maryam Asdjodi-Mohadjer. You provided valuable ideas on the eventual direction of the research as well as feedback that improved the substance of my dissertation. I wish to thank my employer, The Boeing Company, and my immediate management who helped provide a means for this PhD. Thanks to Ed Fleming for reviewing and providing independent feedback of this dissertation. I wish to thank the International Software Benchmarking Standards Group, Limited for the ability to use the Development & Enhancement Repository of project history data. This data provides a wealth of information and should be the instrument for further research in this area. Finally, I am deeply indebted to my family for their continued support and allowed me to pursue my dream. My wife, Helen, who supported me, believed in me, and listened to me. I am especially grateful to her, for without her, I would have never have completed this dissertation. And to Colleen and Ryan, for their patience over the last five years.

Table of Contents

List of Tables

List of Figures

CHAPTER 1. INTRODUCTION

Introduction to the Problem

Software effort estimates are the basis for project bidding, budgeting, planning, and probability estimates (Grimstad, Jørgensen, & Moløkken-Østvold, 2006). These resulting estimates are critical, as poor budgeting and planning often have dramatic consequences (Jørgensen, 2005). When budgets and plans are too pessimistic, business opportunities can be lost, whereas overly optimistic budgets and plans can lead to significant losses during project execution when cost and schedule exceed plans. A lack of detailed information regarding a project's characteristics at the early lifecycle phases leads to difficulty making accurate cost estimates (Berlin, Raz, Glezer, & Zviran, 2009). Research into software project estimation has mainly focused on the practitioners who perform or rely on effort estimation, particularly executives, managers, and technical staff (Kemerer, 1987).

The number of project failures and projects completed over cost and schedule has been a significant issue for software project managers. Although between 30% and 40% of software projects are ultimately completed despite going over budget or schedule, many more projects are cancelled or fail (Moløkken & Jørgensen, 2003). Among the many reasons for failure, inaccuracy in software estimation has been identified as a root cause of a high percentage of failures in the project lifecycle (Jones, 2007; Jørgensen, 2005; Kemerer, 1987; Moløkken & Jørgensen, 2003).

Compounding the challenge of inaccuracy, each new software development introduces a new, unique set of estimation factors to address and problems to solve (Boehm, 2006). The recently introduced agile software development methodology is one area in which research on effort estimation accuracy is scarce (Moløkken-Østvold, Haugen, & Benestad, 2008). To address this research gap, this study also investigates effort estimation accuracy in agile software development methodology.

The process of software project estimation is comprised of a number of steps that each consist of smaller processes (Galorath & Evans, 2006). There are generally two independent activities in software project estimation— estimating size, followed by estimating the development effort based on the size—during which problems can occur in estimating the software project. Thus, software effort estimation is only the first part of the overall estimation process; the effort depending on the estimation size as input. Based on the size of the project, estimation could require from several days to months of team effort. The amount of time and effort necessary cannot be estimated until the total amount of code required for the project can be predicted, which is performed by function point (FP) analysis or direct source line of code (SLOC) estimation based on requirements and experience (Galorath & Evans, 2006).

The process of software effort estimation begins with estimating the code size of the project via SLOC estimation or FP analysis. SLOC estimation is performed using a bottom-up analysis based on previous experience with developing similar systems or specific functionality (Boehm et al., 2000). To enhance the fidelity of the estimate, the SLOC being estimated must be the similar to SLOC with which the estimator has experience. Thus, organizational metric data are useful as a basis of the estimations. FP analysis is the method of measuring the size of a software system as a number of logical function points categorized into one of five types—inputs, outputs, inquiries, internal files, and external interfaces—and assigned both a language independent complexity factor and a value based on number of functions that are estimated (Galorath & Evans, 2006; Larman, 2004).

Accurate prediction of software development continues to challenge software engineering researchers (MacDonell & Gray, 2005). The reasons that software cost estimation is difficult and error prone include (a) software cost estimation requires a significant amount of effort to perform correctly; (b) the process is often done hurriedly, without an appreciation for the effort required to perform the estimate; (c) experience is required for developing estimates, especially for large projects, and; (d) human bias (Agarwal, Kumar, Yogesh, Bharadwaj, & Anantwar, 2001). The software industry is one of the most labor-intensive industries in terms of the human effort required to create a product (Jones, 2007), compounding the estimation problem.

The recently introduced agile development methodology, essentially a hybrid of previous structured development methodologies, is considered a subset of incremental development (Larman, 2004). Agility is often associated with the concepts of nimbleness, quickness, or dexterity (Erickson, Lyytinen, & Siau, 2005). Most agile methods are based on the Agile Manifesto, developed when representatives from lightweight processes such as eXtreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Adaptive Software Development, Crystal, and others met in 2001 to document 12 principles for agile development (Highsmith, 2001).

Historically, software effort estimates have been inaccurate, regardless of the method used, and although agile methodology has recently been embraced by industry, there has been little research conducted into it, especially in the area of estimating agile projects (Erickson et al., 2005).

Background of the Study

Software projects have been plagued by a number of significant problems, including exceeding budget or schedule and not meeting customer requirements, which cause a large percentage of projects to be cancelled well into the development lifecycle. A number of contributing causes to these problems include lack of communication with the customer (especially in requirements understanding), poor development processes such as lack of code inspections or inadequate testing, and the inability to accurately estimate software projects upfront (Boehm, 1981; Jones, 2007).

The annual Standish Group's CHAOS reports provided data indicating very high failure rates of 70% or more and low success rates (Glass, 2006). The 1994 CHAOS report indicated that average cost overrun was as high as 189% (Jørgensen & Moløkken-Østvold, 2006). These reports, considered fundamental to most claims of future crisis, are frequently referred to by researchers in both industry and government (Glass, 2006; Jørgensen & Moløkken-Østvold, 2006). Although these reports may have over-estimated failure rates, the problem of exceeding budget and schedule when conducting software projects is a reality that must be addressed.

There has been little research into the root cause of estimation inaccuracy. One study indicates only a small percentage of research is based on statistical analysis and that the qualitative nature of research can impact the reasons for estimation error (Jørgensen & Moløkken-Østvold, 2004). When Moløkken and Jørgensen (2003) reviewed previous studies, they found that the reasons for the overruns cited are complex and not properly addressed by software estimation surveys. Additionally, Jørgensen and Shepperd (2007) indicated that estimation performance accounted for only 5% of the research topics in their study of 304 software cost estimation papers.

The software-development lifecycle methodology or model plays a role in successful development estimation planning. These methodologies can be categorized into three development lifecycle model categories: the heavyweight group, which includes the classic waterfall and V methodologies; the middleweight group, which includes the incremental and spiral methodologies; and the lightweight group, which includes agile development models such as XP and Scrum (Guntamukkala, Wen, & Tarn, 2006).

The heavyweight group of models began as the first set of lifecycle methodologies. In the 1960s, software development was viewed purely as an art, reflected in the implicit belief in a "code-and-fix" approach to the development process and the need for early establishment of clearly defined, detailed requirements (Guntamukkala et al., 2006). The waterfall methodology, introduced in 1970, was an early attempt to overcome the lack of rigor from the earliest approaches (Royce, 1998). This methodology, comprised of a series of fixed process steps each completed before the next begins (Hoffer, George, & Valacich, 2005), is also known as a sequential approach, referring to the completion of the work within one monolithic cycle (Benediktsson, Dalcher, & Thorbergsson, 2006).

The basis of the waterfall model is composed of two essential steps—analysis and coding. Royce (1970) decomposed the lifecycle into the activities of system requirements, software requirements, analysis, program design, coding, testing, and operations. While waterfall purists indicate that each process step is standalone, the architect of the waterfall model, Winston Royce, reportedly stipulated the need for feedback between the phases (Royce, 1998) to guide the next set of development methodologies.

The waterfall model may best be suited for developing large, complex software because the architecture and functionality is so tightly coupled and integrated that it may not be possible to develop the software incrementally (Turk, France, & Rumpe, 2005). The waterfall model was also introduced to overcome problems encountered in managing large custom software-development projects such as for the U.S. military and has proven to be a successful solution to early problems that had overwhelmed development (Guntamukkala et al., 2006). These traditional heavyweight lifecycle models are best suited for an environment where user requirements, and the technologies needed to meet those requirements, are well understood (Guntamukkala et al., 2006). The waterfall model was refined in the early 1970s to address larger and more complex software projects (Benediktsson et al., 2006). In addition, newer methodologies arose that were categorized by the middleweight lifecycle methodology.

The middleweight lifecycle methodology is composed of models such as the incremental software development model. Incremental development consists of planning, developing, and releasing software products in increments, or phases, whereby each additional increment adds operational functionality or capability not contained in previous releases (Benediktsson & Dalcher, 2004). An increment is defined as “a self-contained functional unit of software with all supporting material such as requirements and design documentation, user manuals and training,” with the first increment often being the core product providing all functionality to address basic requirements (Benediktsson et al., 2006, p. 266). Because it enhances progress by providing finished, operational parts of a system long before the entire system is complete (Guntamukkala et al., 2006), using the incremental model has been recognized as an effective means of maintaining user interest and active involvement in the development of the system. By doing so, it ensures a close fit to real needs and a greater level of user satisfaction (Benediktsson & Dalcher, 2004). Unlike the classic monolithic or ‘big-bang’ approaches typified by the waterfall model, the incremental model is intended to create steadily enhanced versions of a system (Benediktsson, Dalcher, Reed, & Woodman, 2003).

The lightweight lifecycle methodology or agile development methods are recent examples of an approach to incremental delivery but to a much smaller degree of work per iteration (Benediktsson et al., 2003). Agile is described by one of the chief architects of XP as a “lightweight” development methodology for small- to medium-sized teams tolerant of rapid changes in requirements (Beck, 2000, p. xv). Agile methods apply adaptive planning, timeboxed (short discrete time period) iterative and evolutionary development and delivery, through agility (Larman, 2004). The most well-known agile process, XP, includes many of the benefits of incremental development but also encompasses newer development processes (Turk et al., 2005). Agile software development methodology is one area where research on effort estimation accuracy is scarce (Moløkken-Østvold et al., 2008).

As waterfall is historically the predominant development method, especially for larger programs, it was selected for comparison to other development methods. Similarly, incremental development is a middleweight model that is still used with high popularity as indicated by Benediktsson et al. (2003) and is used as a comparative development model. Although new, agile has become a widely adopted development method and because of the sparse amount of research, it was chosen as a comparative method.

Within each development methodology, SLOC or FP can be used for sizing estimation (Boehm et al., 2000; Galorath & Evans, 2006; Jones, 2007; Larman, 2004). In addition, story points are a predominant means of sizing in agile methods (Jones, 2007; Larman, 2004). What may work for sizing estimation in one development method, however, may not work in another.

Again research in the area of sizing methods is light. One study indicates that FP related papers between 2000 and 2004 were only 14% of the total and had decreased 29% from 1990 to 1999 (Jørgensen & Shepperd, 2007).

Software effort estimation can be performed using software models such as the University of California's COnstructive COst MOdel (COCOMO). Estimation models such as COCOMO, Parametric Review of Information for Costing and Evaluation—Software (PRICE-S), and Software Evaluation and Estimation of Resources—Software Estimating Model (SEER-SEM) have been built on a basic functional relationship where the project effort is based on a mathematical relationship of a calibration factor, the software size, a set of scale factors and effort multipliers (Boehm et al., 2000; Carr, 1997). These estimation models use assumptions regarding the project that drive the scale factors and effort multipliers (Boehm et al., 2000). Because the multipliers are based from best guesses or assumptions, incorrect assumptions can lead to large deviations between predicted and actual values when the wrong values are forced to fit the project estimation equation (Menzies, Chen, Hihn, & Lum, 2006). The importance of having accurate or as accurate as possible, size estimate data cannot be overemphasized. The use of poor data will cause significant differences between the model estimate and actual project metrics (Menzies et al., 2006). One study found that the choice of which effort estimation models results in very large performance deviations (Menzies et al., 2006). Therefore, a common aim of effort prediction research is to build and validate models that generate estimates within 25% of the actual effort at least 75% of the time (MacDonell, & Gray, 2005).

A recent version of COCOMO called COnstructive INcremental coCOMO (COINCOMO) has been designed for projects with multiple increments similar to agile development (Boehm & Valerdi, 2008). Although it has been reported that less effort is required with a large number of development increments (Benediktsson et al., 2003), there does not appear to be research on the model's success on agile projects. There are no models developed for agile development (Abrahamsson, Moser, Predrycz, Sillitti, & Succi, 2007).

Statement of the Problem

Inaccurate software effort estimates have plagued software projects for decades. Poor estimates have not only led projects to exceed budget and schedule but also, in many cases, be terminated entirely. The ability to accurately estimate software development projects changes as newer methodologies replace old ones.

Purpose of the Study

The purpose of this study was to determine if there was a difference in software effort-estimation accuracy between the development methodologies of waterfall and incremental methodologies, and the newer agile development methodology. Specifically, the impact of using source line of code (SLOC), function point (FP), or story-point sizing methods were explored across the three common development methodologies of waterfall, incremental, and agile.

Rationale

The number of project failures and projects completed over cost and schedule has been a significant issue for software project managers. A primary root cause of these failures has been inaccurate estimates (Jones, 2007; Jørgensen, 2005; Kemerer, 1987; Moløkken & Jørgensen, 2003). As new development methods such as agile development are introduced, software effort-estimation must be tailored to support the new development method. Research was needed into ways of increasing accuracy for software development organizations. This research was a response to this need and added to the body of knowledge on software estimation.

Based on the problems encountered with software effort estimation, the objective of this research was to identify the impact of using SLOC and FP on estimation accuracy among waterfall, incremental, and agile development methodologies. Additionally, this research directly compared the SLOC, FP, and story-point sizing methods in the agile development methodology. Correlation analysis was used to determine the relationship between project sizing and the estimated project effort to construct the effort cost estimation models.

Research Question

This study addressed the following research question to determine which of the methods investigated yields the highest software estimation accuracy; to what extent does development method and sizing method explain the variability in effort estimation accuracy? This question was explored through a set of hypotheses.

Significance of the Study

This study investigated whether project success was determined by different development and sizing methodologies through a comparison of software effort-estimation accuracies. The basis of the study were data indicating that software effort-estimation inaccuracies led to average effort and cost overruns of between 30% and 40% (Moløkken & Jørgensen, 2003). In addition, there has been a lack of research into software effort-estimation accuracy, especially into the recently introduced area of agile development. This research explored the void in the current body of knowledge regarding the estimation of agile software projects.

Considering the existing literature, it is felt that there was a need to compare different software system development methodologies in terms of their success in software engineering measurements. To fill this gap, SW development methodologies such as agile, waterfall, and incremental were considered. The most common SW effort estimations methodologies such as SLOC, FP and story-point sizing were used to estimate the SW development efforts by agile, waterfall, and incremental methodologies. The comparison of the accuracy of the estimates by these methodologies was the focus of this research.

This study aimed to contribute to a better understanding of software effort-estimation accuracy. Providing better estimates can ensure more realistic estimation of projects allowing managers to make better decisions regarding whether to proceed with projects. Better estimates can also provide for better project execution. Moreover, as more than $300 billion are spent across approximately 250,000 projects on an annual basis, increasing estimation accuracy would save billions of dollars annually (Berlin et al., 2009).

Definition of Terms and Acronyms

Definition of Terms

Agile methodology. A recent software development methodology that encompasses timeboxed iterative evolutionary approaches, adaptive planning, and a number of values and practices that embrace rapid and flexible responses to change.

Effort. The magnitude of the hours and schedule of the product being developed.

Function point (FP) estimation methodology. A software sizing method that determines the size of a software project based on the quantifying functionality from the user's perspective, which in turn is based primarily on logical design.

Incremental methodology. A software development method that consists of the planning, development, and release of software products in increments or by phased development in which each additional increment adds operational functionality or capability not contained in previous releases.

Main Effect. The effect of an independent variable alone on a dependent variable averaged across the levels of other independent variables.

Simple Main Effect. Also known as Simple Effect. The effect of one independent variable within one level of a second independent variable.

Software measure. Any tool that provides a quantitative indication of some attribute of software such as size.

Source lines of code (SLOC) estimation methodology. A software sizing method that estimates the total lines of code in the project. This has historically been the most common software sizing method.

Project effort estimation. A project estimate in person-days or person-months and indicating the effort or resources required to complete a project or phases of the project.

Project schedule estimation. A project estimate in person-days or person-months indicating the calendar time it will take to complete a project or phases of the project based on the number of people in the project.

Project size estimation. An estimate of a project's software size in terms of defined units such as function points or lines of code (LOCs).

Story point. A unit of measure expressing the size of a feature, function, or user story expressed in a numeric point value relative to other features, functions, or user stories.

Waterfall methodology. A sequential approach in which tasks are broken into multiple phases that are all part of one monolithic lifecycle. The waterfall model has been commonly used for managing large custom software development projects.

Acronyms

COCOMO. COnstructive COst MOdel

COINCOMO. COnstructive INcremental coCOMO

COTS. Commercial Off-The-Shelf

DSDM. Dynamic Systems Development Method

DV. Dependent Variable

FP. Function Point

IFPUG. International Function Point Users Group

ISBSG. International Software Benchmarking Standards Group

IV. Independent Variable

KLOC. Thousand Lines of Code

LOC. Lines of Code

OO. Object Oriented

NESMA. Netherlands Software Metrics user Association

PRICE-S. Parametric review of information for Costing and Evaluation - Software

SEER-SEM. Software Evaluation and Estimation of Resources – Software Estimating Model

SLOC. Source Lines of Code

SPSS. Statistical Package for the Social Sciences

UML. Unified Modeling Language

USC. University of Southern California

XP. eXtreme Programming

Assumptions and Limitations

Assumptions

The following assumptions have been made for this study.

1. Software development companies or departments desire to the reduce cost of and time needed for software development.
2. Increased effort accuracy will increase the likelihood of completing projects on time and within budget.
3. The data recorded in the International Software Benchmarking Standards Group (ISBSG) limited database used for the study is random.

Limitations

The following limitations may reduce or negate the generalizability of the findings beyond the present study.

1. The study only collected quantitative data with no corollary subjective data that may add insight into the results.
2. The data collected is limited to what was contributed voluntarily from those who completed projects in the past. Projects in the ISBSG database are from the better-performed part of the industry and therefore don't necessarily represent failures of projects as prevalently as a random sample.
3. The ISBSG data did not have enough samples of incremental development method and story point sizing method.

Nature of Study

This ex-post-facto, quasi-experimental formal study investigated a set of software projects through application of a method of statistical correlation to historical software development project data provided by the International Software Benchmarking Standards Group Limited (ISBSG). This source data represented recent software development projects throughout industries worldwide. Using Statistical Package for the Social Sciences (SPSS) 15.0 software, quantitative statistical techniques were used to analyze the data to determine the accuracy of agile estimation as compared to traditional development methods. The independent variables were the development method (waterfall, incremental, and agile) and sizing method (SLOC, FP, and story-point). The dependent variable was the effort-estimation accuracy.

Organization of the Remainder of the Study

Chapter 1 provides the background of the study and statement of the problem before describing the purpose of the study, research questions, significance of the study, and assumptions and limitations. Chapter 2 provides a thorough review of the literature pertaining to software estimation research, focusing on estimation accuracy and the design of agile practices. Chapter 3 describes the quantitative research design, instrumentation and measures, data collection and analysis procedures, validity and reliability of the research, and ethical considerations. Chapter 4 provides a discussion of the data collection and population, the analysis and the answers to the hypotheses and research question. Chapter 5 provides an overall summary of the research, limitations of the study and recommendations for future research.

CHAPTER 2. LITERATURE REVIEW

Overview of the Software Estimation Problem

The number of project failures and projects completed over cost and schedule has been a significant issue for software project managers for decades (Boehm, 1981; Kemerer, 1987). Specifically, whereas between 30% and 40% of software projects are ultimately completed despite going over budget or schedule, many more projects are cancelled or fail (Moløkken & Jørgensen, 2003).

The Standish Group's annual CHAOS reports have historically indicated that 60 to 80% of projects exceed cost or schedule or fail entirely (Glass, 2006; Jørgensen & Moløkken-Østvold, 2006). However, Glass (2006) argues that the CHAOS reports do not represent actual failures, and are thus being used as sources of fact without validation. In support, Jørgensen and Moløkken-Østvold (2006) alleged that the Standish Group's CHAOS data may be corrupted. Additional studies beyond the CHAOS report surveyed by Moløkken and Jørgensen (2003) have established that a more likely average effort and cost overrun is between 30 and 40%, which still represents a large percentage of projects.

There are a number of reasons for software project failure. Galorath and Evans (2006) conducted an Internet search, which resulted in 2,100 sites indicating over 5,000 reasons why projects fail. Galorath and Evans (2006) reported the most significant reasons include lack of requirements understanding, lack of time or discipline to sufficiently plan the project, and a loss of focus when the project begins. Among the many reasons for failure pointed to by other studies, inaccuracy in software estimation has been identified as a root cause of a high percentage of failures in the project lifecycle (Jones, 2007; Jørgensen, 2005; Kemerer, 1987; Moløkken & Jørgensen, 2003). Stated differently, projects do not fail during implementation but rather during the estimation process within the planning phase (Wells, 1999). Failure to realize a possible error in the initial estimates is a significant contributor to later problems, including overall project failure (Galorath & Evans, 2006).

Software effort estimates are the basis for project bidding, budgeting, and planning (Grimstad et al., 2006). These estimates are performed based on historical knowledge or expertise by a single estimator or by teams (Galorath & Evans, 2006). When budgets and plans resulting from estimates are too pessimistic, business opportunities can be lost because of a high bid, whereas overly optimistic budgets and plans can lead to significant losses during project execution due to cost and schedule overruns (Berlin et al., 2009).

Software effort estimation is the set of techniques and procedures that organizations use to arrive at an estimate for proposal bidding, project planning, and probability estimates (Agarwal et al., 2001; Jørgensen, 2005). As such, estimation accuracy is a very significant issue for executives, managers, technical staff, and, particularly, practitioners who perform or rely on effort estimation (Kemerer, 1987). Accurate prediction of software development effort also continues to challenge software engineering researchers due to the continued lack of accurate estimates (MacDonell & Gray, 2005).

Software cost or effort estimation is not a standalone activity (Jones, 2007). Generally, there are two independent activities in software project estimation; estimation of size followed by estimation of development effort based on the size. Problems can occur in estimating the software project during either of these phases. The development effort cannot be determined until the total amount of code required for the project is predicted based on requirements and experience of the organization (Galorath & Evans, 2006). Software effort estimation is only part of an overall estimation process that depends on the estimated size as input.

There are three broad categories for estimating methods for software projects (Berlin et al., 2009). The first is human expert judgment such as the Delphi method, second is quantitative modeling based on empirical data such as multivariate modeling, and third is machine learning techniques based on artificial intelligence.

The differences in estimating effort when using SLOC, FP, or story point estimation are the basis of this study. SLOC estimation is the oldest metric used for software size as it represents a physical reality (Jones, 2007). SLOC is also the most commonly used due to both manager and developer different interpretations of FP (Sheetz, Henderson, & Wallace, 2009). Although FP estimation has a qualitative aspect in that no two estimates may be the same, it is more suitable for agile development because of the quick method used for size estimation based on features or functionality (Cohn, 2006). The primary argument against using SLOC estimation is that the value will not be known until the software is fully developed, while the primary argument against FP estimation is that it requires a significant investment of time by experts to count system functions (Wu & Kuan, 2008).

These sizing techniques are independent of the development methodology. Classic methodologies, including waterfall and incremental, have used SLOC and FP sizing techniques. With the recent introduction of agile development methodology, however, another challenge for accurate software estimation has been created.

Evolution of Development Methodologies

Classical Development Methodologies

Traditional heavyweight lifecycle models for software development are all consistent with the need for early establishment of clearly defined, detailed requirements (Guntamukkala et al., 2006). The heavyweight models began as the first set of lifecycle methodologies. In the 1960s, software development was viewed purely as an art, reflected in the implicit belief in a ‘code-and-fix’ approach to the development process and the need for early establishment of clearly defined, detailed requirements (Guntamukkala et al., 2006).

When the waterfall methodology was introduced in the 1960s to overcome problems encountered in managing large, custom software-development projects, it proved a successful solution to those early problems that had overwhelmed development (Guntamukkala et al., 2006). This methodology, also known as a sequential approach, is comprised of a series of fixed process steps, each completed before the next begins, until the product within one monolithic cycle is finished (Benediktsson et al., 2006; Hoffer et al., 2005). The waterfall model may be best suited for developing large, complex software projects because their architecture or functionality is often so tightly coupled and integrated that it may not be possible to develop the software incrementally (Turk et al., 2005).

The classical waterfall model was soon refined in the early 1970s to address larger and more complex software projects as major systems were being developed (Benediktsson et al., 2006). As such, the waterfall model has commonly been used for managing large, custom software development projects, such as those for the U.S. military.

The waterfall model, however, is resistant to change, as applying the method requires defining a waterfall style plan and following it throughout the full development project (Keenen, Powell, Coleman, & McDaid, 2006). Thus, traditional heavyweight lifecycle models are best suited for environments where user requirements and the technologies needed to meet those requirements are well understood (Guntamukkala et al., 2006). Whereas waterfall purists indicate that each phase is standalone, the architect of the waterfall model, Winston Royce, reportedly stipulated there needs to be feedback between the phases (Royce, 1998). Evolution of these methodologies continued into the middleweight lifecycle model.

The middleweight lifecycle methodology is typified by the incremental software development model and consists of the planning, development, and release of software products in increments or phased development, whereby each additional increment adds operational functionality or capability not contained in previous releases (Benediktsson & Dalcher, 2004). Benediktsson and Dalcher (2004, p. 5) define incremental development as “the planning, development, and release of software products in increments, where each additional increment adds operational functionality, or capability, not available in previous releases." The first increment is often the core product providing functionality to address basic requirements (Benediktsson et al., 2006). Because it enhances progress by providing finished, operational pieces of a system long before the entire system is complete (Guntamukkala et al., 2006), using the incremental model has been recognized as an effective means of maintaining user interest and active involvement in the development of the system and by doing so, it ensures a close fit to real needs and a greater level of user satisfaction (Benediktsson & Dalcher, 2004). Unlike the classic monolithic or 'big-bang' approaches typified by the waterfall model, the incremental model is intended to create steadily enhanced versions of a system (Benediktsson et al., 2003).

In applying classical development methodologies of the heavyweight and middleweight models, software development has generally followed a proscribed pattern or structure process over the past 40 years (Erickson et al., 2005). The classical methodologies are marketed as being adaptable, yet have often been found so established and full of inertia that they cannot respond sufficiently to a rapidly changing environment to be viable. Normal or classical requirements elicitation has become unsuitable due to rapid changes in competitive threats, stakeholder preferences, development technologies, and time-to-market pressures (Cao & Ramish, 2008). What appeared to be needed, however, was a methodology that could address changing requirements and faster cycle time.

Introduction to Agile Development

The recent introduction of agile development methodology is an instance of a lightweight model although it is considered a subset of incremental development methodology (Larman, 2004). It takes its name in that agility is often associated with the concepts of nimbleness, quickness, or dexterity (Erickson et al., 2005), and agility development is the stripping away of much of the heaviness commonly associated with traditional software development methodologies. It is believed that agile and formal software development are not incompatible but rather can be combined when needed (Turk et al., 2005).

Most agile methods are based on the Agile Manifesto, developed when representatives from lightweight processes such as eXtreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Adaptive Software Development, Crystal, and others met in 2001 to document 12 principles for agile development (Highsmith, 2001). Although, software effort estimates have been inaccurate, regardless of the method used, agile methodology has recently been embraced by industry, but there has been little research conducted into it, especially in the area of estimating agile projects (Erickson et al., 2005).

Agile methods advocate requirements analysis in small steps throughout the development lifecycle (Beck, 1999). As most organizations avoid developing formal specifications or requirement documentation this methodology is gaining use (Cao & Ramish, 2008). Agile software development processes have evolved primarily to support timely and economic development of software that meets customer needs at the time of delivery using development processes that continuously adapt and adjust to (a) the collective experience and skills of the developers, (b) changes in software requirements, and (c) changes in the development and targeted operating environments (Turk et al., 2005). Greater customer satisfaction and more comprehensible requirements evolve from the iterative requirements engineering of agile methods (Cao & Ramish, 2008).

Beck (2000, p. xv) states that XP is a "lightweight" development method that is tolerant of changes in requirements and its four core activities include coding, testing, listening to the customer and to other developers, and designing as an implicit part of the coding process. These four activities are applied to each iteration or build of the development lifecycle.

A fundamental process of agile development is pair programming, a technique in which two programmers work together to develop a single piece of code, which has been shown to yield significantly higher productivity and code quality than is achieved by two programmers working separately (Cohn, 2004). The levels of planning in the agile environment are strategy, portfolio, product, release, iteration, and day planning (Cohn, 2006). For normal project estimation and planning and of relevance to this research are release, iteration, and day planning. Release planning, which occurs at the start of the project and identifies the scope schedule and resources for the project, can be updated throughout the project. Iteration planning, which is performed at the beginning of an iteration based on the content of the previous iteration, identifies the work for the new iteration. Day planning is conducted to coordinate work as well as to synchronize the efforts. All planning is focused on that which will lead to the completion of a task.

Agile Research

There have been a number of studies conducted regarding the effectiveness of agile methods such as XP, with the reported benefits being the ability to work in a greater number of creative, small teams as well as higher user satisfaction levels (Benediktsson et al., 2006). In one recent study, more than 90% of XP projects surveyed claimed to be successful, though this finding was based on responses from developers (Rumpe & Schröder, 2002; Turk et al., 2005). A recent IBM and Sabre Airlines case study focusing on the effects of adopting XP reported an improvement in productivity of 46 to 70% in lines of code per person-month as well as significant decreases in pre- and post-release defect density (Benediktsson et al., 2006). Another study reported that a group of participating students who had no prior knowledge in XP development benefited from the process when performing development (Keenen et al., 2006). Sfetsos, Angelis, and Stamelos (2006) similarly reported that XP practices are easy to apply.

The use of small increments and economies of scale within the development organization that can be adjusted both upwards and downwards have been shown to be key attributes of agile estimation (Banker, Chang, & Kemerer, 1994), supporting Benediktsson et al.'s (2003) conclusion that using a larger number of development increments requires less effort than using one increment as part of a monolithic development cycle.

The social and physical environment has also been studied with respect to agile development. With such physical environment factors as geographic separation, miscommunication and difficulty in establishing contact are likely to increase and be perpetuated by the *telephone-tag* or *e-mail-tag* problem (Turk et al., 2005). One study reported that two major companies, Intel and Hewlett-Packard (HP), discovered agile practices to be valuable in reducing some of the barriers arising from time zone, geographic, and socio-cultural differences (Holmström, Fitzgerald, Ågerfalk, &

Conchúir, 2006). Holmström et al., (2006) also reported agile XP practices improved communication, coordination, and control within global software development teams.

On the other hand studies also indicate social and environmental factors negatively impact agile development. Lee, Delone, and Espinosa (2006) indicate that a conventional agile software development approach like XP has problems that stem from attempting to integrate agile methods to fit geographically distributed software development projects. In particular, separation in time and geographic distance significantly increase the complexity of software development activities, essentially making conventional agile methodology less effective. When compared to same-site work, work across multiple locations takes longer and requires more people for a project of equal size and complexity; indeed, geographic separation can reportedly reduce productivity up to 24% (Javed, Maqsood, & Durrani, 2006). Due to temporal and geographic distance in the global software development realm, key concepts in agile methods are more difficult to realize (Holmström et al., 2006), whereas paired programming exhibits no difference in productivity or quality between co-located or remote locations (Baheti, 2002). Additionally, Turk et al. (2005) indicate that the size of teams can limit the effectiveness and frequency of face-to-face interactions as larger teams focus on larger projects and associated problems.

Turk et al. (2005) reported that another beneficial effect of XP is better understanding between users and developers of each other's problems and needs through close collaboration which impacts efficiency. Supporting Turk et al., informal communications and evolving requirements guiding the project through XP are benefits of face-to-face communications (Cao & Ramish, 2008). Customers can easily use the increments delivered through XP as the basis for determining and verifying project progress while both clarifying and refining requirements (Turk et al., 2005). The frequent delivery of working code provides project visibility whereby the customer can see evidence of an actual product rather than evolving plans in the form of intangible requirements and design documents, which are often presented in terms a customer cannot grasp. On the other hand, it is considered such collaboration is a social activity, and not everyone can learn XP or feel comfortable working within a social environment (Sfetsos et al., 2006).

Evolving requirements is often viewed as an inherent problem in traditional software development (Turk et al., 2005). Requirements can change during software development due to changes in (a) the environment in which the software will be implemented, and (b) the development environment. In contrast, the agile-process community views requirement evolution as an opportunity for developing software that can enhance the customer's competitiveness in a rapidly changing environment.

A significant problem that arises with agile is its reliance on source code as software documentation, which frequently leads to situations in which in-depth knowledge of software products is held only by the developers (Turk et al., 2005). Turk et al. indicate that the reason why documentation as a communication aid is de-emphasized is based on an XP assumption that implicit knowledge has greater value than externalized knowledge; it has even been suggested that design activity may no longer be a requirement. However, in reality design activity is deeply integrated into XP processes (Succi & Marchesi, 2001). This less formal rigor within agile methodology's XP processes supports a lower effort estimation and actual effort.

Turk et al. (2005) proposed that the code is the most accurate and reliable description of what a system does and how it was designed. As the system is defined by the code, the code describes reality. Based on this argument, agile and XP methods provide all the documentation required to satisfy the most formal of projects while not requiring extra administrative effort. Studies have shown that a significant amount of the effort required to evolve systems is spent understanding the code (Turk et al., 2005) and that releasing a product to a customer for maintenance without adequate classical documentation may require additional effort.

Although agile methodology specifically targets small- and medium-sized projects, there have been arguments for scaling agile and XP processes to large projects (Turk et al., 2005). In their study, Lindstrom and Jeffries (2004) also found that XP has emerged as an alternative to comprehensive methods designed primarily for very large projects. Large-scale agile development can be accomplished through the use of smaller *hub-teams* that interact differently than that of sub-teams in traditional hierarchical organizations (McMahon, 2006).

Proponents claim that XP's unique composition of best practices and omission of time-intensive software-engineering activities can help downsize larger projects (Turk et al., 2005). They also note that agile processes can be extended to address XP limitations. Boehm (2006) proposes a hybrid agile and document-driven approach that would be well suited for large programs.

Project visibility can be achieved solely through the delivery of working code. The short iteration lengths in agile methodology facilitate timely customer feedback that helps ensure that the end product will meet customer needs at the time of delivery (Turk et al., 2005).

With the benefits and popularity of agile development, organizations need a process to create accurate estimates.

Software Estimation Process

History

The practice of software-development effort estimation began in the 1950s with the development of a simple manual rule-of-thumb process for project planning (Jones, 2007). With the increased and widespread use of computers in the 1960s, the number and size of software projects grew, creating a need for the development of technology for formalized software estimation (Jones, 2007). As early as 1965, software estimation models were available and being used (Boehm & Valerdi, 2008). These early estimation models were based on a linear function of size, modified by a complexity factor of the function (Boehm & Valerdi, 2008). Several of the early developers in the 1960s include the well known names of Joe Aron of IBM and Dr. Barry Boehm and Larry Putnam of Intel (Jones, 2007).

In the late 1970s, estimation models evolved to be based on simple effort complexity factors that were often subjective and their simplicity did not provide the accuracy that estimations required (Boehm & Valerdi, 2008). IBM conducted much of the research in and development of software estimation. The first automated estimation tool, the Interactive Productivity and Quality (IPQ) tool, was written by Capers Jones and Dr. Charles Turk at IBM in 1973 (Jones, 2007). Based on the work that continued in the

1970s, an additive, exponential, and multiplicative parametric model emerged that served as the basis for the 1981 COCOMO model, the general form of which is the following (Boehm & Valerdi, 2008, p. 75):

$PM = A * (SIZE)^{B} * (EM)$, in which

- PM is the project effort in person-months.
- A is the calibration factor.
- SIZE is a software module's functional size, expressed in lines of code.
- B is the combined set of scale factors that have an exponential effect on software development effort.
- EM is the combined set of effort multipliers that have a multiplicative effect on software development effort.

This equation became the foundation for software estimation models.

In his 1981 book *Software Engineering Economics*, Dr. Barry Boehm discussed various software cost estimation algorithms that are still referenced in a number of software estimation research papers today (Agarwal et al., 2001; Jørgensen & Moløkken-Østvold, 2004; Jones, 2007). Boehm's COCOMO discussed in the book is the predominant model used both in research and industry due to its open nature and free right to use (Jones, 2007).

Other software estimation models developed in the 1980s were commercial or proprietary although derived their form from COCOMO (Boehm & Valerdi, 2008). These models, which include the Parametric Review of Information for Costing and Evaluation—Software (PRICE-S), Software Evaluation and Estimation of Resources—Software Estimating Model (SEER-SEM), and Putnam Software LIfecycle Management

(SLIM) models, have continued to evolve over the last 25 years and are still being offered today.

The most significant change in software estimation models during the 1980s was the introduction of FP sizing (Boehm & Valerdi, 2008) after the development of the FP metric by Allan Albrecht and colleagues at IBM (Jones, 2007). This metric is based on five external attributes of software applications: (a) inputs, (b) outputs, (c) inquires, (d) logical files, and (e) interfaces.

The increased number of new development methodologies beginning in the late 1990s, such as Object Oriented (OO), Unified Modeling Language (UML), and application generators, created the need to update the COCOMO model to the COCOMO II, which was released in 2000 (Boehm & Valerdi, 2008). It also led to the development of specific COCOMO model spin-offs, such as Constructive Incremental COCOMO (COINCOMO), that are used for incremental development.

The Software Estimation Process

A project estimation process described by Galorath and Evans (2006) is comprised of the following steps:

- Establish the estimate scope.
- Establish the technical baseline ground rules and assumptions.
- Collect the data.
- Size the software.
- Prepare a baseline estimate.
- Quantify risks and conduct a risk analysis.
- Review, verify, and validate the estimate.

- Generate a project plan.
- Document the estimate and lessons learned.
- Track the project throughout development.

The first three steps of this process involve gathering the data required to make the size estimate, with the process of sizing the software immediately followed by estimation of the effort. Only the first five steps are relevant and explored in this study.

Based on the size of the project, estimation could require from several days to months of team effort. The process of software effort estimation begins with estimation of the code size of the project, which can be accomplished via SLOC estimation or FP analysis. SLOC estimation is performed using a bottom-up analysis based on previous experience with developing similar systems or specific functionality (Boehm et al., 2000, p.14).

Using the SLOC estimation method provides a concise, quantitative indication of the true size of the software that can be easily compared to what was actually built (Galorath & Evans, 2006). Function points, on the other hand, are both a quantitative and qualitative functional units of measure for the project software from the user's perspective. FP estimation can be performed when the system functionality is known, but is more resource intensive and requires more training and time than SLOC estimation. FP analysis is performed by assessing the functions that the system executes according to the five user function types—internal logical files, external interface files, external inputs, external outputs, and external inquiries—then applying a complexity factor which produces the unadjusted function point count (Boehm et al., 2000). A team of senior developers should be the source that performs FP estimation so that consistency and historical experience are applied (Galorath & Evans, 2006). Using the FP method, the conversion to SLOC can be performed manually (Boehm et al., 2000) or by estimation tools such as COCOMO (Boehm et al., 2000) and SEER-SEM (Galorath & Evans, 2006) based on programming language and other driving parameters. However, effort estimation accuracy does not appear to improve when formal models are used, and projects estimated with FP analysis have had larger overruns than projects estimated by other methods (Moløkken & Jørgensen, 2003). Nevertheless, function counts are likely to be sufficiently accurate to be of use to software managers, having been found to have a 75.1% correlation with SLOC estimation (Kemerer, 1987). While the cost of the estimation tools is comparatively low compared to the monetary risk of inaccurate estimates, there is a cost in person-hours to learn to use these tools.

Agile development estimation introduces story points that are in some ways similar to function points. Story points are functional stories or use cases of the system though are sized relative to each other (i.e. story A is twice as big as story B) and assigned an abstract value (Cohn, 2006). Expert opinion, analogy, or disaggregation are used to arrive at the estimates of the story points. Once the size of the code base is projected, the effort required by the team organization can be estimated. Using COCOMO, SEER-SEM, or many of the other modeling tools can provide the effort estimate based on parameters adjusted for the organization and project.

Lewis (2001) reports that there is debate regarding the ability to apply mathematical and scientific principles to software estimation between the process camp (engineering) and the problem-solving camp (art) for software estimation. If software estimation is a rigorous engineering process analogous to building a house, then estimations should be more accurate. In the case of houses or large structures, engineering tools are available to accurately estimate the materials and labor provided. Software estimation, on the other hand, is an art, and if the initial educated guess of software size is inaccurate, then the resulting effort and schedule estimates for the software development project will be correspondingly inaccurate (Carr, 1997). Lewis (2001) notes that the fundamental issue is not the cost or effort model rather it is the software size or complexity estimate. Most estimations of effort are still based on expert judgment, as no significant empirical evidence supports using formal estimation models (Jørgensen, 2005). If historical data for use as bases for code estimates are unavailable, a size estimate is an educated guess.

Guidelines for software-development effort estimation based on experience and empirical research differ from mainstream guidelines in the following ways (Jørgensen, 2005):

- They base estimates on expert judgments rather than models.
- They are easy to implement.
- They use the most recent findings regarding judgment-based effort estimation.

Schedule estimation is not as efficient as effort estimation because schedule slippages are primarily due to causes extraneous to projects (Carr, 1997). As Moløkken and Jørgensen (2003) point out, few researchers have conducted extensive analysis of the reason behind effort and schedule overruns.

A review of the literature indicates that there are insufficient data for any new assessment, especially for assessment of a new development methodology such as agile methodology. This is confirmed in one paper that hypothesizes that there is a fundamental factor in effort estimation—effort itself—that has precluded comparative assessment and led effort estimation models to suffer from very large performance deviations (Menzies et al., 2006).

A common aim of effort prediction research is to build and validate models that deliver estimates within 25% of the actual effort and within 75% of schedule (MacDonell & Gray, 2005). Every modeling framework makes assumptions that, if incorrect, can lead to the wrong equations being forced to fit the project data, and therefore, large deviations between predicted and actual values (Menzies et al., 2006). The significance of having an accurate, or as accurate as possible, size estimate data cannot be overemphasized.

A way to reduce deviations is to reduce the number of variables in a model, which decreases the deviation of a linear model. If the large deviation between the actual results and the model cannot be resolved, then the general experts or model-based effort estimation group cannot quantitatively assess the merits of different supposedly best practices (Menzies et al., 2006).

Learning an effort estimation model is easier when the user is not required to fit the model to "noisy" project data (i.e., when the project data contains spurious information not associated with variations among projects; Menzies et al., 2006, p. 887). Menzies et al. note noise can come from many sources, including clerical errors or missing variable values.

The percentage of error is relative to the actual effort. Organizations that wish to use algorithmic estimating tools must collect historical data on their own projects in order to calibrate the models for local conditions, thereby increasing their accuracy (Kemerer, 1987).

Effort and schedule estimates are derived from the estimated size because the effort required to build a software system is specified as a function (Carr, 1997). Thus, in the simplest form:

Where S is a measure of the system size and E is effort, then f(S) = E

Exploring the function's basic structure, it has been found that:

$E = (a + bSc)mX$, in which m(X) are multipliers (based on known factors that influence the effort) and a, b, and c are constants.

The size of the project is also a major factor in the fidelity of the effort estimate. For systems less than 20K LOC, actual effort tends to be either accurately predicted or underestimated. For systems greater than 20K LOC, estimations of effort are generally overestimated, and this overestimation becomes progressively larger as software size increases (Carr, 1997).

COCOMO Model Estimation Tool

COCOMO was originally developed by Barry Boehm in 1981 and extensively revised as COCOMO II in 2000 (Boehm et al., 2000). Because many tools have been based on the COCOMO (USC, 2002), there is little variability among the tools. COCOMO assumes that effort increases more than linearly as software size increases (Menzies, Port, Chen, Hihn, & Stukes, 2005). For example, a 20,000-SLOC project requires more than four times the effort as a 5,000-SLOC project.

COCOMO is built around a basic functional mathematical model that has tens of parameters defining the characteristics of the software development that affect the effort and are input that changes the weights of scale factors or effort multipliers (Boehm et al., 2000). Though the parameters are converted into numeric values, qualitative research as to why a person, for example, selected *low* for the documentation match parameter rating when the criterion was *some lifecycle needs uncovered* (Boehm et al., 2000) needs to be addressed. From another perspective, the weighting scale for each parameter is based on user input or from historical data; therefore, investigating why a value assigned to *low* is 1.2 versus 1.1 is also important in researching the accuracies of software project estimates.

The core intuition behind COCOMO-based estimation is that as a program increases in size, the development effort increases exponentially. More specifically (Menzies et al., 2006, p. 884):

$$\text{Effort (in person-months)} = a * (\text{KLOC}^{b}) * \left(\prod_{i=1}^{n} EM_i\right)$$

The effort rises exponentially from b, which are the scale factors relating to the economies or diseconomies of scale. EM is the effort multiplier and contains 22 parameters that must be tuned to the specific project (Boehm et al., 2000).

Software effort estimation models should be calibrated to local data from incremental holdout studies, defined as studies that determine the utility of the calibrated parameters using data not used during calibration, combined with randomization and hypothesis testing and repeated a statistically significant number of times (Menzies et al., 2005).

Annual releases of COCOMO to the public with more accurate parameter calibrations are contingent on the continued addition of historical projects to the

COCOMO II calibration database (USC, 2002). Noting that the benefit of participating in data collection is the availability of a more accurate predictive model for estimating software project costs, USC requests that users complete cost estimate questionnaires. This could lead to a considerable research database.

USC has complemented COCOMO with COCOTS, COSYSMO, and COINCMO. COCOTS, a member of the USC COCOMO II family of cost estimation models developed for estimating the expected initial cost of integrating Commercial Off-The-Shelf (COTS) software into a new software system development or system refresh, currently focuses on three major sources of integration costs (Yang, Boehm, & Clark, 2006). While the COCOTS model was developed with the intention of determining the economic feasibility of COTS-based solutions, its model structure consists of most, if not all, of the important project characteristics that should be carefully examined when assessing COTS-based development project risk (Yang et al., 2006).

Software Estimation Research

Much of the literature on software-development effort estimation is based on research conducted from the early 1980s through the early 1990s (Kemerer, 1987; Moløkken & Jørgensen, 2003). Current research has tended to concentrate on other unique research dimensions and topics of software estimation, or, as in the case of Boghossian (2002), analysis of software estimation data from previous results. Estimating effort on the basis of expert judgment remains the most common approach today because no significant empirical evidence supports using formal estimation models (Jørgensen, 2005). Moreover, debate continues regarding whether software estimation should be considered an engineering practice in which rigorous processes are applied or an art in which accuracy is dependent on skill (Lewis, 2001), lending credibility to qualitative data. When software cost estimates are conducted early in the software development process, the estimate may be based on wrong or incomplete requirements.

A survey of studies reveals that quantitative research remains the predominant research methodology (Carr, 1997; Jørgensen & Shepperd, 2007; Liu & Mintram, 2006; Moløkken & Jørgensen, 2003), with few qualitative studies having been conducted in this area. Because quantitative methods use actual numerical data, cost estimation error is relatively easy to measure in quantitative research. However, the measured data can be difficult to interpret properly when attempting to determine why the measure of actual effort does not reflect the estimate (Grimstad & Jørgensen, 2006).

Kemerer's (1987) quantitative study of the accuracy of effort estimation led him to conclude that further research is required to develop understanding in this area and to recommend addressing how the productivity of software developers could be improved qualitatively. Software effort estimation could be conducted using a mixed-methods approach whereby the collection of qualitative data supplements the collection of quantitative data. Using an open-ended question form in a survey can provide for the exhaustive research capability unlike a closed-question form that contains simply a weighted category of *other* or *don't know* (Cooper & Schindler, 2006). Additionally, using the qualitative data in a mixed-method approach provides both inductive and deductive research processes (Creswell & Creswell, 2005).

In their study, Grimstad and Jorgensen (2006) found that an analysis framework could be applied to effort estimation research, providing that the 'how' and 'why' of following software estimation processes are captured through qualitative analysis. Support for the framework can be derived from the position that because software engineering is human-resource intensive, it is important to understand and evaluate the value of different types of experiences and their relationship to the quality of the developed software (Wohlin, 2004). Cooper and Schindler (2006) support Wohlin's perspective in that an appropriate use for qualitative analysis is process understanding. Therefore, software project estimation processes can be analyzed through qualitative methods.

Similarly, supporting a qualitative or mixed method in human intuition is often unreliable in assessing the consequences of managerial intervention, such as adjusting staff level on a complex software-development process (Abdel-Hamid, Sengupta, & Swett, 1999). The results of many such interventions were reported as "surprising" by the participants, defending the argument that both quantitative and qualitative methods are required for this data analysis (p. 534).

Risk analysis is often performed in conjunction with effort estimation using qualitative data according to quantitative criteria (Mizuno, Kikuno, Takagi, & Sakamoto, 2000). Addressing the risk parameter in the estimation through a qualitative method provides a means of handling the qualitative nature of the project's risk assessment. By using a mixed method, the study becomes a full analysis of how the risk values affect the estimate as well as how those values were derived.

However, researchers must remain aware of the limitations posed by the social aspects of qualitative data. When Wohlin (2004) attempted to evaluate whether straightforward quantifiable measures were sufficient to gain understanding of the performance of individuals, he found that he had to consider the large differences between individuals. Wohlin (2004) indicates that given the results, it would have been valuable to complement quantitative results with a qualitative evaluation.

One driving factor that leads to the suitability of qualitative methods is the process that managers or engineers use in assigning the proper values for the parameters in estimating the project. Because the factors of software size and complexity are based on expert judgment, the estimation model may be characterized as a combination of a formal quantitative model and expert judgment (Grimstad & Jørgensen, 2006). The general goal of Grimstad and Jørgensen's (2006) analysis was to acquire more knowledge of the estimation ability (performance) of two different estimation methods: the estimation model and expert estimation. They note that despite the availability of this estimation model, most projects have been estimated by unsupported expert-judgment based estimation, which supports a more qualitative approach to effort estimation.

In their survey study, Moløkken and Jørgensen (2003) reported that respondents believed that cost overruns were most often caused by over-optimistic estimates (51%), closely followed by changes in design or implementation (50%). The reason for schedule overruns were optimistic planning (44%) followed by frequent major changes from the original specifications (36%). Thus, both cost and schedule overruns have similar root causes. They also note expert judgment-based estimation methods are the most frequently used methods, a possible cause of estimation errors.

In their assessment of 10 prior studies, Moløkken and Jørgensen (2003) found that significant differences in methodology, as well as the broad span in years among the studies, made any correlation of study results all but impossible. They also argued that the lack of procedures ensuring random sampling and the lack of proper analysis of the samples may have been major problems with all of the surveys, potentially leading to difficulties when interpreting and transferring results. Jick (1979) argued that because qualitative data are predominantly superior to quantitative data in clarity of meaning, qualitative data could have provided these authors with a better means of correlating the studies.

The top-down and bottom-up approaches are two primary heuristic approaches used to perform software project estimation (Agarwal et al., 2001). Top-down estimation includes historical comparisons, design-to-cost specifications, and quick-and-dirty estimates. MacDonell and Gray (2005) point out that because informal estimation methods based on experience, expert opinion, or personally moderated analogies are still commonly used, it would be reasonable to consider the issue of industry acceptability before greatly advancing a new estimation method. Similarly, they indicate a significant majority of estimates are based on *estimation by analogy*, an expert-judgment estimation method found by one Dutch study to be used in 62% of estimations. This estimation method relies heavily on experience in and knowledge of similar development environments and historically maintained databases on the accuracy of completed projects (Agarwal et al., 2001). As did Agarwal et al. (2001), Liu and Mintram (2006) found that software-project effort prediction models are often based on historical data sets.

The accurate prediction of software development effort continues to challenge researchers. Several factors lead to inaccuracy, such as noisy, incomplete, inaccurate, and inconsistent data and the lack of a method that takes into account all the variables and their impact (MacDonell & Gray, 2005). Noisy data can be the result of a lack of validity and/or reliability, the impracticality of the measurement tool, and raw errors in the data (Cooper & Schindler, 2006). Research supports that both quantitative and qualitative factors are important in addressing the challenge of noisy data (MacDonell & Gray, 2005).

MacDonell and Gray (2005) concluded that the estimation tasks performed by project managers early in a development process would benefit from a strategy that utilizes fuzzy logic models in a manner complementary with other algorithmic estimation approaches. This strategy provides a range of predictions as opposed to a single point value, reducing or removing the unwarranted level of certainty associated with a point estimate. MacDonell and Gray asserted that the heuristic approximation concept of fuzzy logic supports qualitative input values and can generate quantitative outputs for the effort estimate using a stored fuzzy rule base. However, the researchers found that despite the significant research that has been conducted over the prior two decades, project managers continue to prefer and use ad-hoc estimation methods based on personal experience, expert opinion, and local analogies.

Jørgensen and Shepperd's (2007) systematic review of 304 journal articles regarding software cost estimation in 76 journals uncovered that a small percentage (<10%) of the articles related to the use of qualitative methods. Abdel-Hamid et al. (1999) investigated the impact of different project goals on software project planning and resource allocation decisions and project performance. They conducted their research within the context of a simulation game in which quantitative data for the simulation was the quantitative output and participant behaviors the qualitative output. Abdel-Hamid et al. justified their method by explaining,

> Since actual events on a software project almost always differ from the assumed events that the plans were designed to meet, project managers must react continuously to real world events that actually occur, and not to those that might have occurred (p. 533).

The use of qualitative methods in PhD research on software project estimation is similarly lacking. In his dissertation, *An Investigation into the Critical Success Factors of Software Development Process, Time, and Quality,* Boghossian (2002) investigated a wide range of factors affecting software development, including the estimation of the project. His statement of the problem was that "software projects take too long, run over budget, do not meet customer requirements and in many cases are cancelled well into the development lifecycle (p. 3)." He conducted interviews with 11 software experts across 10 companies over a period of 18 months. Boghossian found that all of his research questions, such as, "What strategies might be or have been effective in reducing or eliminating software project failures?" could all be answered using qualitative methods (Chapter 4).

In their study investigating the reasons for estimation errors, Jørgensen and Moløkken-Østvold (2004, p. 1006) asked of which differences in the reported types of reasons for estimation error are dependent on the analysis method, i.e., whether applying a qualitative analysis of project experience reports or a quantitative (statistical regression) analysis of the same project data? Based on their finding that subjects reported indirect causal reasons much more frequently in interviews than in project-specific estimation experience reports, they proposed qualitative interviews with senior personnel should be conducted to determine the reason for estimation error.

To examine the impact of the role of the respondent, Jørgensen and Grimstad (2008) focused their data collection approach and analysis techniques on collecting reasons for estimation error based on approaches in the same organization, conducting general interviews with eight employees responsible for the estimates and 68 project estimation experiences. They found that one source of inaccurate estimates is irrelevant and misleading information based partly on unconscious effects. They also found that client expectations and word variations in work description also strongly affect the estimates.

When estimating effort, three actions must be performed

- Ensure everyone involved in the estimation clearly understands that the purpose is to derive the most likely use of effort.
- Exclude estimators that access misleading or irrelevant information that can bias the estimates.
- Exclude estimators with vested interests in the outcome.

Problems with data collected from actual industry projects can have an impact on the data quality. Using better instruments or more experienced researchers to collect the data can increase the accuracy. Adding more values to increase the fidelity to a category of *other* in a survey allows the researcher to uncover what *other* really means through the qualitative method. In one noted study, the reasons for estimation inaccuracy were described as factors not controlled by the respondent (Jørgensen & Moløkken-Østvold, 2004).

One of the fundamental issues is the difference between the size of the estimated software and the as-built software. On average, the size of the actual implemented software deviates ±33.4% from its initial size estimation (Carr, 1997). Because some of the growth is related to changes in customer or user requirements, removing any requirement changes would uncover the true deviation from baseline estimation to completion.

Grimstad et al. (2006) take a deterministic view (estimates as one single effort value) instead of a probabilistic view (estimates as a combination of effort value and probability) of effort estimation. This is due to the fact that software development organizations nominally do not treat estimation as a separate activity but rather an integrated part of project planning, pricing, and budgeting. The authors question if the term *effort estimate* is precisely defined and whether estimates and actual effort are comparable when evaluating estimation accuracy. In order to examine possible reasons for the lack of precise estimation terminology in the software industry, they reviewed the actual use of estimation terminology. They found deficiencies in its use across academia and industry that they suggest be remedied through the use of common software development tools. Rule (2000) also supports the probabilistic view but notes that the human preference for a single value forces a point estimate.

Other research is geared towards enhancing the existing tool algorithms. Menzies et al. (2005) calibrated effort estimation models using an extensive search over the space of calibration parameters in an estimation model. Their technique is proposed to be much simpler than other effort estimation methods yielding similar predictive accuracy levels because it requires fewer project data and attributes, but much research is still required.

Additionally, the highly variable reaction of project personnel to new technologies makes the software development process uncontrollable (MacDonell & Gray, 2005). There have been no further studies with respect to this model.

Research is leading to the evolution of standard models and tools. Choi and Sircar (2006) found that although the most commonly cited measure of software size is FP analysis, it lacks applicability to the new technology environment and has some significant measurement problems. They developed a Product Points Model (PPM) that can be used very early in the lifecycle, when requirements are immature to accurately estimate effort. However, effort estimation accuracy did not improve when formal models were used and projects estimated with FP analysis had larger overruns than other projects (Moløkken & Jørgensen, 2003).

Another factor that has been addressed is integrating COTS software with developed code as another means of development, especially with the large number of applications and libraries available. Yang et al. (2006) found that the use of COTS products in COTS-based development (CBD) brings with it a host of unique risks quite different from those associated with in-house developed software. The nature of depending on other software in CBD often places software processes and products under greater uncertainty. This lower fidelity of the estimate increases the risk that the project will go over cost and schedule. With the increase in COTS integration regardless of the development methodology used, research in this area will become more important.

Software effort estimation can also be performed during later lifecycle phases. Software development and software maintenance are two different activities with different characteristics; whereas the focus of software maintenance is changing the existing software, the focus of software development is creating new software (De Lucia, Pompella, & Stefanucci, 2002). Hoffer et al. (2004) reported that the cost of system maintenance now accounts for 60% to 80% of the system's lifecycle. This is supported by De Lucia et al. (2002) in making the estimation and planning of software maintenance work a key factor in a successful maintenance project.

Finally, Devnani-Chulani (1999) investigated the improvement in the accuracy of a software project cost estimation model using a Bayesian approach against prior projects' calibration data. The researcher found that the Bayesian approach improves estimate accuracy and has high internal validity but, like other methods, requires further research and verification.

Agile Software Development Estimation Research

Because agile methods have only been in existence over this last decade, research and information on the suitability of agile processes is often based on anecdotal accounts (Turk et al., 2005). The body of research into agile modeling appears to be even sparser than that for extreme programming (Erickson et al., 2005). Abrahamsson, Moser, Pedrycz, Sillitti, and Succi (2007) similarly indicate that no specific effort estimation models have been developed for agile or incremental development.

As Buglione and Abran (2007) note, planning and scheduling processes have been tailored to agile methodologies, but there has been much less focus on the estimation process. One study indicated that group predictions or estimations were less optimistic when agile's planning poker estimation technique was used supporting the case that more accurate estimates can be made by individuals (Moløkken-Østvold et al, 2008).

The research regarding the estimation process for agile development has been positive. Abrahamsson et al. (2007) obtained favorable results when they applied their proposed incremental, iteration-based prediction model to two agile projects. Likewise, Robiolo and Orosco (2007) reported an improvement in effort estimation when using an alternative sizing technique similar to use cases but composed of only transaction and entity objects.

Summary

Development models can be categorized into three model groups: heavyweight which includes the classic waterfall and V models; middleweight, which includes the incremental and spiral models, and; lightweight which includes agile's, XP and Scrum (Guntamukkala et al., 2006). Benediktsson et al., (2006) also indicated the waterfall model has been commonly used for managing large custom software development projects such as those for the U.S. military. Incremental development methodology is "a self-contained functional unit of software with all supporting material such as requirements and design documentation, user manuals and training," and has become a popular methodology; recognized as an effective means of maintaining user interest and active involvement in the development of the system (Benediktsson et al., 2006, p. 266; Guntamukkala et al., 2006). Finally, agile development has become highly used within the last decade focusing on the delivery of business value in small fully integrated releases (Benediktsson et al., 2006; Turk et al., 2005). Each of these models has benefits and drawbacks that must be considered. For example, Turk et al., (2005) note the waterfall model may best be suited for developing large, complex software because the architecture or functionality is so tightly coupled and integrated that it may not be possible to develop the software incrementally. The literature reviewed can be summarized by Benediktsson and Dalcher (2004), Benediktsson et al. (2006), and Merisalo-Rantanen, Tuunanen, and Rossi. (2005), as the use of suitable lifecycle model (exhibiting a specific degree of flexibility) will lead to successful software projects that are delivered on time within budget and to the customer's satisfaction. Another important factor is project managers can customize software lifecycle models to a specific degree of flexibility by combining features of models within each category, creating a model that is most suitable for their project situations (Guntamukkala et al., 2006).

Also based on the literature review, SLOC, FP, and story point sizing methods are common methods used in the estimation process (Cohn, 2006; Galorath & Evans, 2006; Jones, 2007). These sizing methods are used in different development methodologies though story point is part of the agile development. As research in the area of sizing methods is light, especially, in the agile methodology, which has only been in existence over this last decade, (Turk et al., 2005), it is hoped this study can add to the body of knowledge in this area.

CHAPTER 3. METHODOLOGY

The purpose of this study was to determine if there was a difference in software effort-estimation accuracy between the classical development methodologies of waterfall and incremental methodologies, and the newer agile development methodology. Specifically, the impact of using source line of code (SLOC), function point (FP), or story-point sizing methods were explored across three common development methodologies of waterfall, incremental, and agile. Analysis of the International Software Benchmarking Standards Group (ISBSG) database was conducted using Statistical Package for the Social Sciences (SPSS) 15.0 software and examined the relationship between the independent variables (IVs) (estimation parameters) and the dependent variable (DV) estimate accuracy of agile projects.

The following research question was addressed:

To what extent does development method and sizing method explain the variability in estimation accuracy?

The research question was addressed by the following hypotheses:

H1: There will be a main effect for development method on effort-estimation accuracy.

H1o: There will be no main effect for development method on effort-estimation accuracy.

H2: There will be a main effect for sizing method on effort-estimation accuracy.

H2o: There will be no main effect for sizing method on effort-estimation accuracy.

H3: There will be development method by sizing method interaction effect on effort-estimation accuracy.

H3o: There will be no development method by sizing method interaction effect on effort-estimation accuracy.

H4: There will be a simple main effect for development method at each level of sizing method.

H4o: There will be no simple main effect for development method at each level of sizing method.

H5: There will be a simple main effect for sizing method at each level of development method.

H5o: There will be no simple main effect for sizing method at each level of development method.

While acknowledging that many factors may influence the project size estimate, this study focused on the development method used, which is the first independent variable. The second independent variable is the sizing method. The first hypothesis compared only the development method IV on effort-estimation accuracy. The second hypothesis compared only the sizing method IV on effort-estimation accuracy. The third hypothesis correlated the relationship of both sizing methods and development method IVs to effort-estimation accuracy. The fourth hypothesis explored the effect of sizing method on development method. Finally, the fifth hypothesis explored the effect of development method on the sizing method.

Research Design

This study used a quasi-experimental method to analyze the project data contained in the ISBSG database. It employed SPSS software to analyze the results and evaluate the hypotheses. The results were provided through statistical tests. These tests identified the relationship to the hypotheses. Research utilized a 3 x 3 Factorial analysis of variance (ANOVA) to consider all levels of all factors (IVs). The research also employed one-way ANOVA to explore the IVs independently.

Justification for Approach

Creswell (2003) explained, “The quantitative method is one in which the researcher primarily uses postpositivist claims for developing knowledge . . . employs strategies of inquiry such as experiments and surveys, and collects data on predetermined instruments that yield statistical data” (p. 18). Quantitative techniques are well suited for studying large samples such as obtained by the ISBSG database, and generalizing the results from the sample under study to a population. Quantitative research can be experimental or non-experimental and in the form of quasi-experimental, correlational, or descriptive research. Quantitative research can be used to address the core characteristics of description, comparison of groups, association, prediction, and explanation (Holton & Burnett, 2005). However, in order for the researcher to provide an answer through the core characteristic, statistical tools are used to analyze the data.

Quantitative methods differ from qualitative methods in that the former attempts precise measurement whereas the latter seeks to develop understanding for further interpretation (Cooper & Schindler, 2006). Cooper and Schindler (2006) also indicate that the sample sizes differ between the two methods, with quantitative methods normally applied to large sample sizes such that assumptions can be made for a population.

For this research, a pure qualitative methodology would not provide the ability to explore differences between estimates and actual project data.

Although researchers using a quantitative method thoroughly understand the target of their research, researchers using a qualitative method may not fully understand what is being addressed, and therefore sense themes or use inductive reasoning or interpretation to generate meaning (Ruona, 2005).

Qualitative research can have serious weaknesses and problems. Collecting and analyzing the data is a highly labor-intensive endeavor that often generates much stress, even for an experienced research staff (Miles, 1979). Qualitative fieldwork is traditionally demanding, and the increasing demands of collecting and analyzing qualitative data tend to completely overload the researcher throughout the process. The most serious and central difficulty in the use of qualitative data is that methods of qualitative analysis are not well formulated (Miles).

Quantitative project data was captured from both the estimates and the actual project metrics from the database. Explanatory and correlational hypotheses were evaluated using project estimation and actual project metric data consisting of the variables of effort, personnel used, cost, and schedule.

Sample

The sample for this research was selected from the Version 11 of the ISBSG database, which contains data on 5,052 software projects. The target population was the set of projects voluntarily contributed in the database. The sampling method used by the ISBSG was convenience sampling and the questions were strictly voluntary. Selection of the sample was based on the validity rating provided by the ISBSG in the two data quality-rating fields for each set of data. Specifically, each set of data is rated according to the following metrics:

- A: Data submitted were assessed as sound with no factors affecting their integrity.
- B: Data submitted appear fundamentally sound but several factors could affect their integrity.
- C: Because significant data were not provided, it was not possible to assess the integrity of the submitted data.
- D: Due to one factor or a combination of factors, the submitted data have little validity.

Data sets assigned a value of *D* were excluded from the study. Data sets assigned a value of *A* were included and data sets assigned a value of *B* and *C* were evaluated based on their significance to the study.

Instrumentation/Measures

A representative Data Collection Form used by the ISBSG (International Software Benchmarking Standards Group Limited [ISBSG], 2008) was developed by the

International Function Point Users Group (IFPUG). This form collects data in the form of responses to 141 questions in the seven areas consisting of Submitter Information, Project Process, Technology, People and Work Effort, Product, IFPUG or Netherlands Software Metrics User Association (NESMA) Project Function Points, and Project Completion. There are four different questionnaires used: Data Collection Questionnaire New Development, Redevelopment or Enhancement Sized Using IFPUG Or NESMA Function Points; Data Collection Questionnaire New Development, Redevelopment or Enhancement Sized Using COSMIC Function Points; Data Collection Questionnaire New Development, Redevelopment or Enhancement Sized Using Mk II Function Points, and; Data Collection Questionnaire New Development, Redevelopment or Enhancement Sized Using Other Methods. Minor differences in the number of questions between these questionnaires are based on the specific method used.

The data collection form is used by the ISBSG to correlate and report project data software-development estimation models.

Data Collection

The data collection process for this study was conducted using a convenience sample obtained through voluntary submittal of project data to the ISBSG. This researcher obtained a copy of the ISBSG database as the source for the research. Version 11 of the ISBSG database, which was released in 2009, contained data on 5,052 projects. Data for those projects identified as using waterfall, incremental, and agile methodologies as well as those sizing methods of SLOC, FP and story point, were analyzed.

Data Analysis

Data was collected from the ISBSG database and imported into the SPSS database. A 3 x 3 factorial analysis of variance (ANOVA) was conducted to test the hypotheses against the data in the database as presented in Figure 1. The analyses included a frequency analysis cross-tabulation and, one-way ANOVA (Table 1). The results of the analysis are presented in Chapter 4, Tables 6 through 11. The data analysis began with decoding the responses associated with the project's record. Appendix A presents the list of the ISBSG fields used. The data was rated for quality, with outliers based on linear regression and correlation analysis filtered beyond what the ISBSG reported as valid data. One or more outliers in a dataset will increase the value of the standard error of estimation (Berlin et al., 2009). The collected data was imported into the SPSS database. Field and variable definitions were verified. A new variable —Effort Accuracy Error —was created for each record based on the following formula:

Effort Accuracy Error % = ((Actual Effort – Estimated Effort)/ Estimated Effort) * 100

If the Effort Accuracy Error % is positive the actual effort was more than estimated. If negative, the actual effort was less than estimated.

Questionable data and duplicate cases were removed based on the data-quality field. Frequency analyses were generated to assess the overall data sample. Each hypothesis was tested against the project data. Data confidence at the 95th percentile cutoff (P<= 0.05) was performed then compared against the null hypothesis. The results were documented and interpreted.

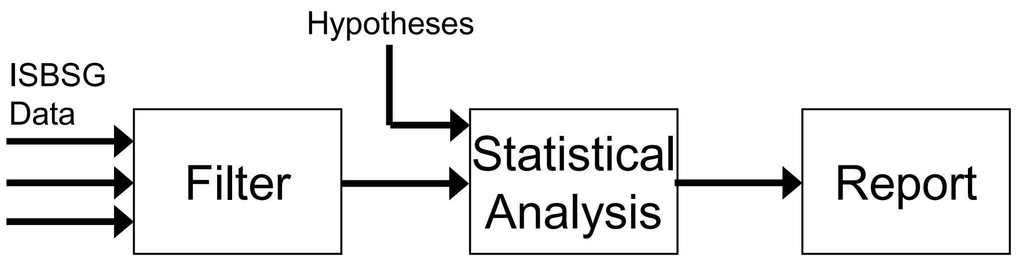

Figure 1. Conceptual framework of the research.

Table 1.
Association of Statistical Analysis to Research Question and Hypotheses

Research Question and Hypotheses	Variables	Statistical Analysis
To what extent does development method and sizing method explain the variability in estimation accuracy?	<u>Dependent</u> Effort Estimation Accuracy (%) <u>Independent</u> Development Method Sizing Method	frequency analysis One-way ANOVA
H1. There will be a main effect for development method on effort-estimation accuracy H2. There will be a main effect for sizing method on effort-estimation accuracy H3. There will be development method by sizing method interaction effect on effort-estimation accuracy H4. There will be a simple main effect for development method at each level of sizing method H5. There will be a simple main effect for sizing method at each level of development method	<u>Dependent</u> Effort Estimation Accuracy (%) <u>Independent</u> Development Method Sizing Method	3 X 3 factorial ANOVA

The ISBSG source data/media was destroyed at the end of the research as stipulated by the research user agreement. Because these data are available from the ISBSG, only the SPSS-processed results output will be retained for two years.

Validity and Reliability

Data validity was determined by the quality rating field assigned to the data sets in the ISBSG database. The ISBSG database has been successfully used by a number of researchers as well as by industry, either by direct purchase or with estimation tools such as SEER-SEM.

A measurement is reliable to the degree that it supplies consistent results (Cooper & Schindler, 2006). Both internal and external validity must be considered. Internal validity is the degree to which an instrument measures what it is designed to measure, which can be affected by threats of history, maturation, testing, instrumentation selection, statistical regression, and experimental mortality (Cooper & Schindler, 2006). For this research, the threat of history was mitigated because the projects being measured are considered single events. Maturation due to the function of the passage of time during the data collection was mitigated as the reporting is obtained through a single questionnaire. The threat of testing was mitigated because actual project data rather than experimental data were analyzed. Instrumentation threat was low because of actual data being reported though it was limited to collection from the instrument and not followed-up qualitatively. Selection threat is a risk that must be addressed. As the population providing the data on the project questionnaires did so voluntarily, a percentage of failures may not have been reported. Statistical regression threat was not a factor because all data was addressed. Attrition from the threat of experiment mortality was not a factor because the data addressed this threat.

External validity where the observed causal relationship is generalized across times, settings, and persons (Cooper & Schindler, 2006) and was not a factor because no experimentation was conducted in this research.

A test performed to examine data validity as recommended by the ISBSG (2010) was a normalization ratio calculated from the Normalised [sic] Work Effort divided by Summary Work Effort. The Normalised Work Effort included estimated hours for effort not completed at the time of data submittal and the Summary Work Effort is actual work that was completed at the time of the questionnaire. Therefore, a ratio of exactly 1.0 indicated there was no estimated effort remaining. A ratio larger than 1.0 indicated some level of estimated work remained on the project when submitted. The increase in estimated data increased the risk to data validity. As a Normalization Ratio greater than 1.20 had a of higher risk data validity problems, records where ratios that were greater than 1.20 were examined using the Project Activity Scope field to ensure the projects were through the development phase of the lifecycle which reduced the risk.

Ethical Considerations

Ethical considerations were minimized due to the quantitative nature of the study and lack of direct involvement in the data collection process. However, this did not mean that ethical considerations were not considered entirely. The data were only used for academic research and destroyed at the completion of the research, per ISBSG stipulation that media obtained for the purpose of research is destroyed after the research is completed. The data did not contain any proprietary or personal information because the ISBSG ensures no data of this nature was included in the project records in the database. This study was submitted to the Institutional Review Board (IRB) for review and approval prior to conducting the research.

CHAPTER 4. RESULTS

The results of the data analyses used to describe the sample and address the research question are presented in this chapter. The data analyses are divided into three sections. The first section uses cross tabulations and measures of central tendency and dispersion to provide a profile of the projects. The second section describes analysis of variance performed on the independent variables and dependent variable. The statistical analyses related to the research question and hypotheses are included in the third section.

Research Overview

Problem Statement

Inaccurate software effort estimates have plagued software projects for decades. Poor estimates have not only led projects to exceed budget and schedule but also, in many cases, be terminated entirely. The ability to accurately estimate software development projects changes as newer methodologies replace old ones.

Purpose

The purpose of this study was to determine if there was a difference in software effort-estimation accuracy between the classical development methodologies of waterfall and incremental methodologies, and the newer agile development methodology. Specifically, the impact of using source line of code (SLOC), function point (FP), or story-point sizing methods were explored across the three common development methodologies of waterfall, incremental, and agile. Analysis of the International Software Benchmarking Standards Group (ISBSG) database was conducted using Statistical Package for the Social Sciences (SPSS) 15.0 software and examined the relationship between the independent variables (IVs) (estimation parameters) and the dependent variable (DV) estimate accuracy of agile projects.

Research Methodology

This study used a quasi-experimental method to analyze the quantitative project data contained in the ISBSG database. It employed SPSS 15.0 software to analyze the results and evaluate the hypotheses. The results were provided by statistical tests of the data. The hypotheses and the research question were explored through these tests. Research employed a 3 x 3 Factorial analysis of variance (ANOVA) and considered all levels of all factors (IVs). Appendix B contains the SPSS syntax code used for the analysis.

Data Collection and Population

Demographic Characteristics of the Sample Data

Data for this study came from revision 11 of the ISBSG project database. In total, there were 5,052 unique project records in the database. The project data records spanned from 1989 to 2009 and included projects from many countries and covered a large spectrum of industries such as aerospace, automotive, government, insurance, manufacturing, medical, and transportation.

Out of the 5,052 projects, 613 projects (12.1%) were used for this study. The reasons for the smaller sample were:

- Only projects where waterfall, incremental, and agile development methods were used.
- Only those projects where SLOC, FP, and story point sizing methods were used.
- The supplied Data Quality Rating was used for determining the reliability of the data for the study. In the projects where the Data Quality Rating was rated by the ISBSG as *D*, the data could not reliably be used for the study because of one or more factors.
- Valid analysis for this study was based on the existence of a calculated DV - Effort Accuracy Error (%). Only those projects where Normalised [sic] Work Effort and Effort estimate fields were present were used so a valid DV could be calculated.

Data Analysis

Cross Tabulations and Central Analysis

Table 2 presents the overall sample frequencies across the independent variables Development Method and Sizing Method, and the dependent variable, Effort Accuracy Error (%). Of the total (*N* = 613) projects indicated in the Effort Accuracy Error % column, 13% (*N* = 79) contained valid Development Method data and 31% (*N* = 187) contained valid Sizing Method data. Note the inter-quartile ranges between 25% and 75% fall within -0.447 and 46.424.

Table 2.
Frequency Counts of Valid and Missing Independent Variables Development Method and Sizing Method and the Dependent Variable Effort Accuracy Error (%)

		Development Method	Sizing Method	Effort Accuracy Error (%)
N	Valid	79	187	613
	Missing	534	426	0
Variance		.416	.103	26756.248
Minimum		1	1	-88.0
Maximum		3	3	2523.1
Percentiles	25	1.00	2.00	-.447
	50	1.00	2.00	14.054
	75	1.00	2.00	46.424

A cross tabulation was generated and presented in Table 3. There were a total of eight possible combinations that could be produced by the IVs. Five of the eight possible combinations were not available. Incremental development method and the story point sizing method were not present. There was also no combination of SLOC and Agile methods. In addition, the three valid combinations only contained one instance.

Table 3.
Cross Tabulation of Development Method and Sizing Method

		Sizing Method		Total
		SLOC	Function Point	
Development Method	Waterfall	1	1	2
	Agile	0	1	1
Total		1	2	3

Table 4 presents the detail of IV development method frequency counts by the three development method types. Of the total number valid records 87.3% (*N* = 69) were of the Waterfall development method, 1.3% (*N* = 1) were Incremental and 11.4% (*N* = 9) were Agile. With only one instance of Incremental method present, testing the hypotheses was impossible.

Table 4.
Development Method Frequency by Development Type

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Waterfall	69	11.3	87.3	87.3
	Incremental	1	.2	1.3	88.6
	Agile	9	1.5	11.4	100.0
	Total	79	12.9	100.0	
Missing	System	534	87.1		
Total		613	100.0		

Table 5 presents a similar account detailing the IV sizing method. Of the total number of records with a valid sizing method, 10.7% (*N* = 20) were of the SLOC sizing method, 88.8% (*N* = 166) were Function Point and 0.5% (*N* = 1) were story point. Again, with only one instance of story point method identified, testing the hypotheses was impossible.

Table 5.
Sizing Method Frequency by Sizing Type

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	SLOC	20	3.3	10.7	10.7
	Function Point	166	27.1	88.8	99.5
	Story Point	1	.2	.5	100.0
	Total	187	30.5	100.0	
Missing	System	426	69.5		
Total		613	100.0		

Analysis of Variance on Independent Variables

A one-way ANOVA was performed individually on the IV Sizing Method effect on Effort Accuracy Error (%) and for the IV Development Method effect on Effort Accuracy Error (%).

Table 6 presents the Mean Effort Accuracy Error (%) for each of the three sizing methods as well as mean for all three methods. As there was only one story point project (*N* = 1), the mean (*M* = 185.034) was skewed higher as compared to SLOC (*M* = 4.891) and FP (*M* = 48.324) while no standard deviation could be calculated.

Table 6.
Mean and Standard Deviation of Effort Accuracy Error (%) by Sizing Method

Sizing Method	*M*	*SD*	*N*
SLOC	4.891	40.8702	20
Function Point	48.324	105.0410	166
Story Point	185.034	.	1
Total	44.410	101.2246	187

In spite of the high mean value of story point, as presented in Table 7, there was no difference in the mean Effort Accuracy Error (%) for the three sizing method types as the significance was greater than the cutoff (Sig = 0.073).

Table 7.
Tests of Between Subject Effects for Sizing Method

Source	Type III Sum of Squares	*df*	*MS*	*F*	Sig.	Partial Eta Squared
Corrected Model	53554.287(a)	2	26777.144	2.660	.073	.028
Intercept	53751.349	1	53751.349	5.339	.022	.028
Sizing_Method	53554.287	2	26777.144	2.660	.073	.028
Error	1852281.434	184	10066.747			
Total	2274646.004	187				
Corrected Total	1905835.721	186				

Note a: R^2 = .028 (Adjusted R^2 = .018)

Table 8 represents the estimated marginal means of Effort Accuracy Error by Sizing Method. Note the extremes of the 95% Confidence Interval for Story Point (Lower Bound = -12.917, Upper Bound = 382.986). This was due to the single story point project in the data. Also of note, the means for SLOC (*M* = 4.891) and FP (*M* = 48.328) are positive indicating the effort estimations tended to be lower than actual effort.

Table 8.
Estimated Marginal Means of Effort Accuracy Error (%) by Sizing Method

			95% Confidence Interval	
Sizing Method	*M*	Std. Error	Lower Bound	Upper Bound
SLOC	4.891	22.435	-39.373	49.154
Function Point	48.324	7.787	32.960	63.688
Story Point	185.034	100.333	-12.917	382.986

Figure 2 graphically represents the data in Table 8 presenting the estimated marginal means for the sizing method. Again, it is noted the increase of the story point marginal means due to the single instance of story point sizing method in the data.

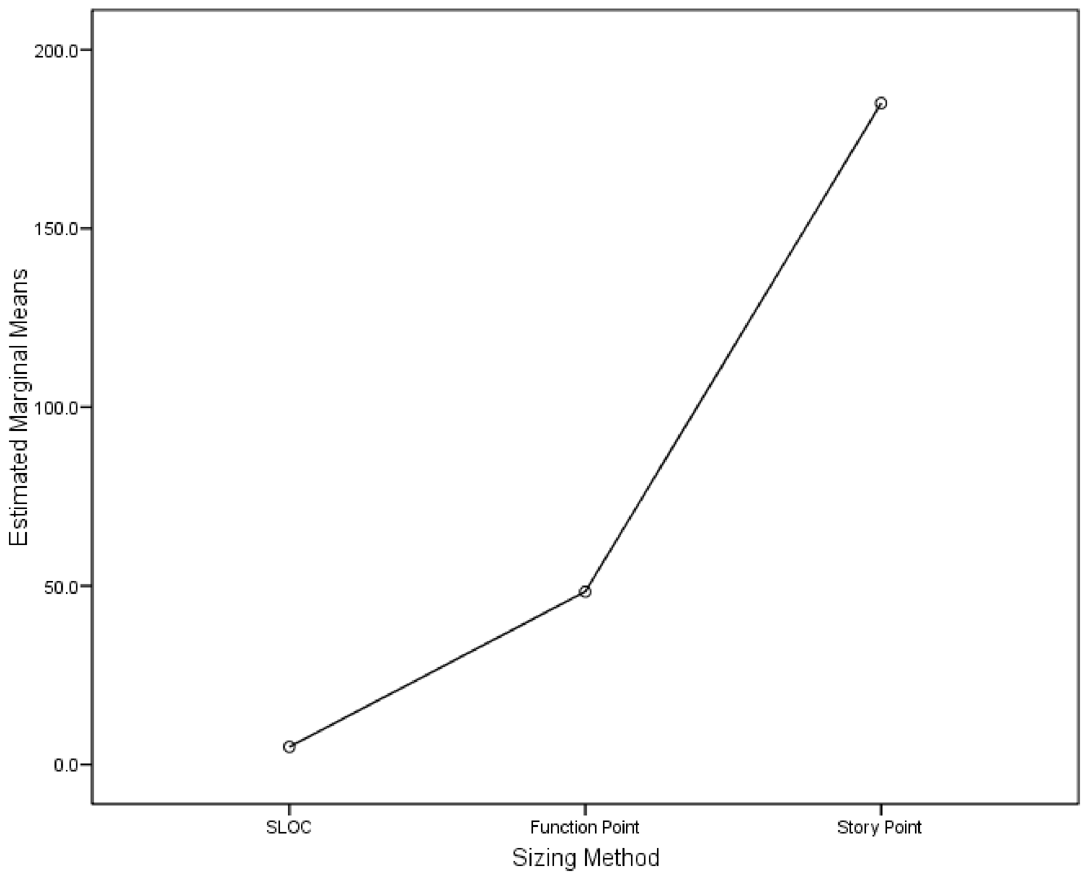

Figure 2. Estimated marginal means of effort accuracy error (%) by sizing method.

Table 9 presents the Effort Accuracy Error (%) for each of the three development methods. As there was only one project with the incremental development method (*N* = 1), the mean (*M* = 155.583) was skewed as compared to waterfall (*M* = 22.103) and agile (*M* = 48.171) and no standard deviation could be calculated.

Table 9.
Mean and Standard Deviation of Effort Accuracy Error (%) by Development Method

Development Method	*M*	*SD*	*N*
Waterfall	22.103	67.1077	69
Incremental	155.583	.	1
Agile	38.171	60.6366	9
Total	25.623	67.4451	79

Notwithstanding the high mean value for the single occurrence of incremental, as presented in Table 10, there was no difference in the mean Effort Accuracy Error (%) for the three development method types as the significance was greater than the cutoff (Sig. = 0.121).

Table 10.
Tests of Between Subject Effects for Development Method

Source	Type III Sum of Squares	*df*	*MS*	*F*	Sig.	Partial Eta Squared
Corrected Model	19161.714(a)	2	9580.857	2.169	.121	.054
Intercept	41394.939	1	41394.939	9.373	.003	.110
Development_Method	19161.714	2	9580.857	2.169	.121	.054
Error	335648.361	76	4416.426			
Total	406676.859	79				
Corrected Total	354810.076	78				

Note a: R^2 = .054 (Adjusted R^2= .029)

Table 11 represents the estimated marginal means of Effort Accuracy Error (%) by Development Method. Note the extremes of the 95% Confidence Interval for Incremental (Lower Bound = 23.224, Upper Bound = 287.942). This was due to the single incremental project in the data. Also of note, the means for waterfall (Mean = 22.103) and agile (Mean = 38.171) are positive indicating the effort estimations tended to be lower than actual effort.

Table 11.
Estimated Marginal Means of Effort Accuracy Error (%) by Development Method

			95% Confidence Interval	
Development Method	*M*	Std. Error	Lower Bound	Upper Bound
Waterfall	22.103	8.000	6.169	38.037
Incremental	155.583	66.456	23.224	287.942
Agile	38.171	22.152	-5.949	82.291

Figure 3 graphically represents the data in Table 11 showing the estimated marginal means for the development method. Again, it is noted the increase of the incremental marginal means due to the single instance of incremental development method in the data.

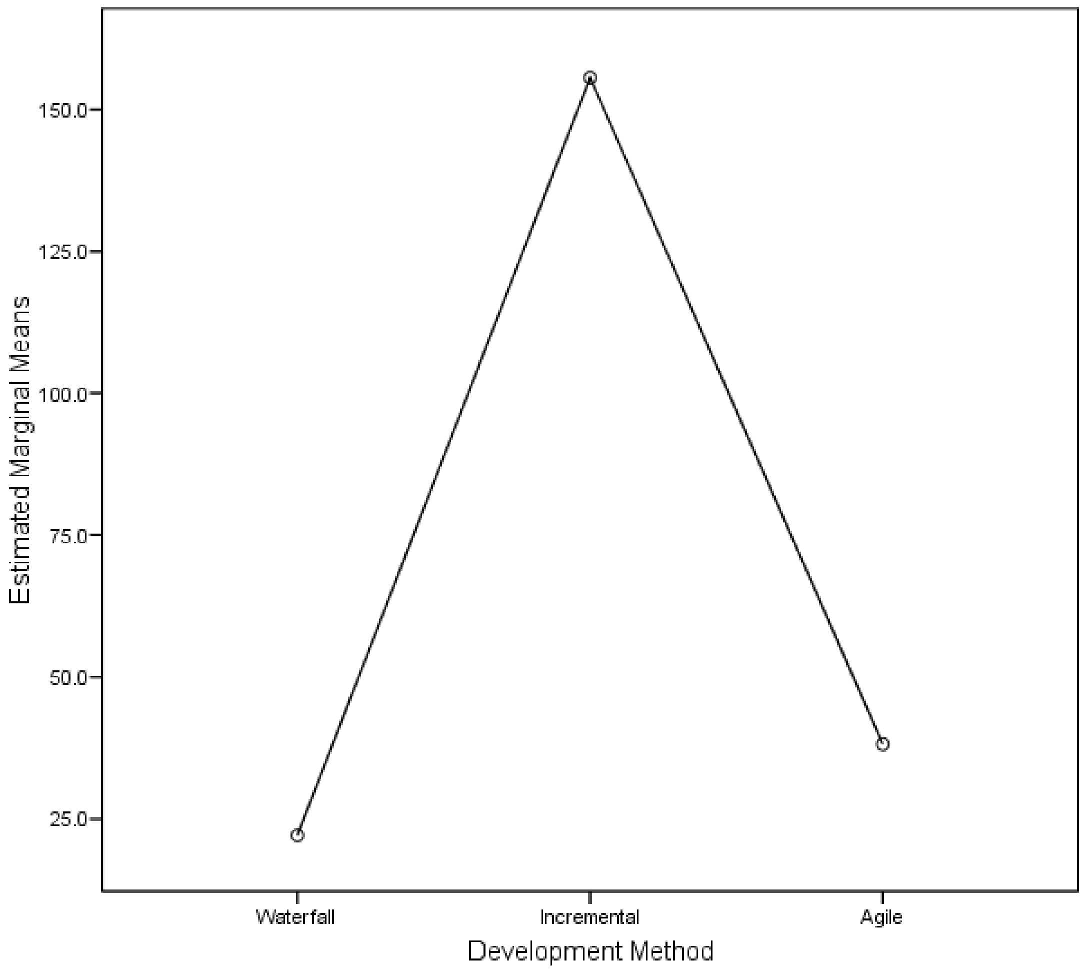

Figure 3. Estimated marginal means of effort accuracy error (%) by development method.

Figure 4 is a scatter plot of the effort accuracy error (%) for all data points. This also shows how the Effort Accuracy Error (%) was relatively flat across all the projects supporting the analysis that there was no difference in the Effort Accuracy Error (%) for either of the IVs.

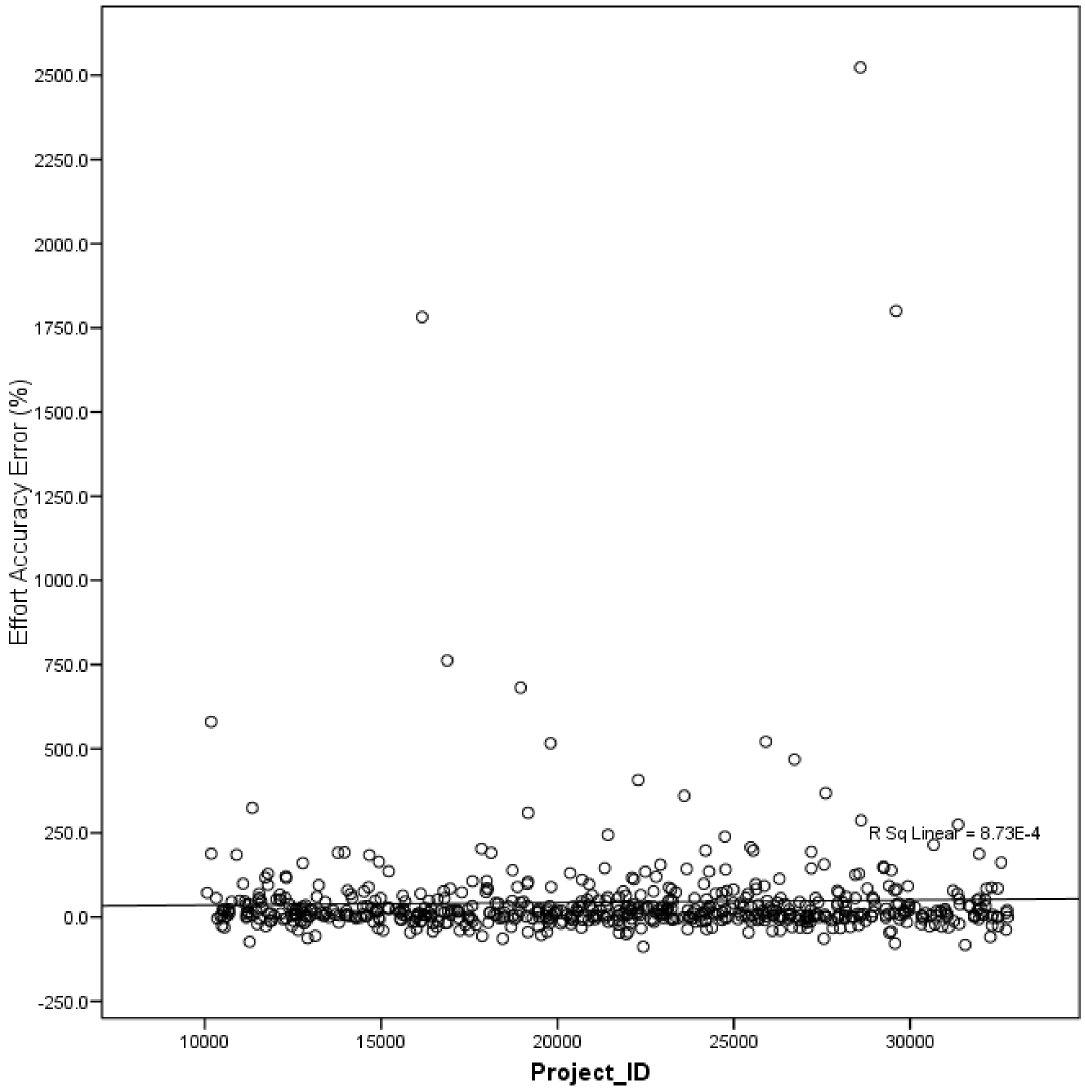

Figure 4. Scatterplot of effort accuracy error (%) for projects used.

All SPSS code used for this study is provided in Appendix B in the form of an SPSS syntax file.

Hypotheses Testing

Five hypotheses were derived from the research question, "to what extent does development method and sizing method explain the variability in estimation accuracy?"

To test the hypotheses, the study employed a 3 x 3 factorial analysis of variance (ANOVA) to consider all levels of all factors (IVs) addressing each hypothesis.

An attempt to conduct this analysis in SPSS generated no usable information. Again, there was not enough data needed to perform the statistical analysis.

Table 12 presents a matrix view again indicating there were only three IV combinations or intercepts that could be analyzed and no factorial analysis could be performed.

Table 12.
Estimation Coefficients for Development Method and Sizing Method

Parameter	Development Method			
	Waterfall		Agile	
	Sizing Method		Sizing Method	
	SLOC	Function Point	SLOC	Function Point
(Intercept)	1	1	1	1
[Development_ Method=1]	1	1	0	0
[Development_ Method=3]	0	0	1	1
[Sizing_Method=1]	1	0	1	0
[Sizing_Method=2]	0	1	0	1
[Development_ Method=1] * [Sizing_ Method=1]	1	0	1	0
[Development_ Method=1] * [Sizing_ Method=2]	0	1	0	0
[Development_ Method=3] * [Sizing_ Method=2]	0	0	0	1

Hypothesis 1. Hypothesis 1 (null) stated that there will be no main effect for development method on effort-estimation accuracy. Due to the insufficient valid number of cases there was insufficient evidence to conclude a difference exists for main effect for development method on effort-estimation accuracy.

Hypothesis 2. Hypothesis 2 (null) stated that there will be no main effect for sizing method on effort-estimation accuracy. Due to the insufficient valid number of cases there was insufficient evidence to conclude a difference exists for main effect for sizing method on effort-estimation accuracy.

Hypothesis 3. Hypothesis 3 (null) stated that there will be no development method by sizing method interaction effect on effort-estimation accuracy. Due to the insufficient valid number of cases there was insufficient evidence to conclude a development method by sizing method interaction effect on effort-estimation accuracy.

Hypothesis 4. Hypothesis 4 (null) stated that there will be no simple main effect for development method at each level of sizing method. Due to the insufficient valid number of cases there was insufficient evidence to conclude a simple main effect for development method at each level of sizing method accuracy.

Hypothesis 5. Hypothesis 5 (null) stated that there will be no simple main effect for sizing method at each level of development method. Due to the insufficient valid number of cases there was insufficient evidence to conclude a simple main effect for sizing method at each level of development method.

Research Question. This research proposed to answer the question; to what extent does development method and sizing method explain the variability in effort estimation accuracy? Though the question could not be answered through the hypotheses from the available data, the question was explored using one-way ANOVA of the IVs independently. As was shown in Tables 7 and 10, the significance of the differences in the means of Effort Accuracy Error (%) by sizing method (Sig. = 0.073) and the means of Effort Accuracy Error (%) by development method (Sig. = 0.121) were greater than cutoff. As there was only one instance of the story-point sizing method and one instance of the incremental development method, there were not enough samples to conclude whether these methods were statistically the same or different. Based on the high significance, however, of Effort Accuracy Error (%) between SLOC, and FP sizing methods, there was no difference between these methods. There was also no statistical difference in Effort Accuracy Error (%) between waterfall, and agile development methods.

Summary of Data Collection and Analysis

Of the 5,052 projects, 613 (12.1%) were used for this research. The data for the study produced only three combinations for the 3 x 3 factorial ANOVA which did not cover all eight possible combinations. Additionally, for each of the three combinations, there was only one instance populated. Thus, statistical analysis for each of the hypotheses could not be performed.

Though the hypotheses were not able to be tested, the research question was explored through separate one-way ANOVAs for each IV. In the test of Effort Accuracy Error (%) for the three sizing methods, there was no difference in the Effort Accuracy Error (%) for SLOC and FP. As there was only one instance of the story-point sizing method, Effort Accuracy Error (%) could not be evaluated though the data point fell within the statistical range of the other two methods. In the test of Effort Accuracy Error (%) for the three development methods, there was no difference in the Effort Accuracy Error (%) for waterfall and agile. Similarly, as there was only one instance of the incremental development method, Effort Accuracy Error (%) could not be evaluated though this data point also fell within the statistical range of the other two methods.

CHAPTER 5. DISCUSSION, IMPLICATIONS, RECOMMENDATIONS

This chapter is organized in four sections. The first section summarizes the purpose of the study, the research questions, hypotheses, and methodology. The second section discusses the results, including answers to the research questions. The third section presents the conclusions, covering both the meaning of the results on academic research as well as software and management practitioners. The final section examines the limitations of the study and proposes recommendations for future research.

Research Summary

Review of Research Purpose

The purpose of this study was to determine if there was a difference in software effort-estimation accuracy between the development methodologies of waterfall and incremental methodologies, and the newer agile development methodology. Specifically, the impact of using source line of code (SLOC), function point (FP), or story-point sizing methods were explored across the three common development methodologies of waterfall, incremental, and agile. Analysis of the International Software Benchmarking Standards Group (ISBSG) database was achieved using Statistical Package for the Social Sciences (SPSS) 15.0 software which examined the relationship between the independent variables (IVs) (estimation parameters) and the dependent variable (DV) estimate accuracy of software projects.

Review of Research Question

This study addressed the following research questions to determine which of the methods investigated yielded the highest software estimation accuracy: To what extent does development method and sizing method explain the variability in effort estimation accuracy?

Review of Research Hypotheses

The research question was composed of and explored through the following hypotheses:

H1: There will be a main effect for development method on effort-estimation accuracy.

H1o: There will be no main effect for development method on effort-estimation accuracy.

H2: There will be a main effect for sizing method on effort-estimation accuracy.

H2o: There will be no main effect for sizing method on effort-estimation accuracy.

H3: There will be development method by sizing method interaction effect on effort-estimation accuracy.

H3o: There will be no development method by sizing method interaction effect on effort-estimation accuracy.

H4: There will be a simple main effect for development method at each level of sizing method.

H4o: There will be no simple main effect for development method at each level of sizing method.

H5: There will be a simple main effect for sizing method at each level of development method.

H5o: There will be no simple main effect for sizing method at each level of development method.

Review of Research Methodology

This study used a quasi-experimental method to analyze the quantitative project data contained in the ISBSG database. It employed SPSS 15.0 software to analyze the results and evaluate the hypotheses. The results were provided by statistical tests of the data. The hypotheses and the research question were then investigated through these tests. Research attempted a 3 x 3 Factorial analysis of variance (ANOVA) that considered all levels of all factors (IVs). One-way ANOVA was also conducted on each IV to answer the research question. Appendix B contains the SPSS syntax code used for the analysis.

Results

Based on the research methodology described, the ISBSG project database was tested using frequency distribution and central analysis, one-way ANOVA, and through a 3 x 3 factorial ANOVA for the hypotheses. Finally, the research question was addressed based on the results. This section summarizes these results.

Hypothesis Testing

Surprisingly, the number of combinations between the IVs, and the single occurrence of the valid combinations, were not enough to support the 3 x 3 factorial ANOVA.

Answer to the Research Question

This research proposed to answer the question, "to what extent does development method and sizing method explain the variability in effort estimation accuracy?" Though the question could not be answered through testing of the hypotheses from the available data, the question was answered by independently addressing the two IVs—development method and sizing method. The differences in the effort accuracy error (%) by sizing method had a significance greater than the cutoff (Sig. = 0.073) indicating that there was no difference between SLOC and FP sizing methods. As there were not enough statistical samples of story point, that method's effect could not be determined. Similarly, there was also no difference in the effort accuracy error (%) between waterfall, and agile development methods (Sig. = 0.121). There were not enough statistical samples of incremental, however, to determine effect.

Implications for Practitioners

Indeed, from the perspective of software development management, the results of this study contradict some long-held views, but also provide some surprises which may affect the way practicing managers want to plan for software development programs. Though industry has moved from development method to development method (where now agile is the latest to be embraced), advocates have claimed that each new development method is "better" than the previous. Nonetheless, the overall performance of the methods may still be debatable, but the accuracy of effort estimation is not. Though there was limited data for the study, there appeared no statistical difference in estimation accuracy. As this was similarly true for the three sizing methods, practitioners should feel free to use any of these methods when estimating projects. Additionally, based on the data used, the research indicated that estimates are generally lower than actual suggesting that practitioners factor this bias into their projects.

Conclusions

This study set out to explore the effect of the IVs development method and sizing method on effort estimation accuracy.

The literature review derived that development models can be categorized into three model groups: heavyweight which includes the classic waterfall and V models; middleweight, which includes the incremental and spiral models, and; lightweight which includes agile's, XP and Scrum (Guntamukkala et al., 2006). Each of these models has benefits and drawbacks that must be considered. For example, Turk et al., (2005) noted the waterfall model may best be suited for developing large, complex software because the architecture or functionality is so tightly coupled and integrated that it may not be possible to develop the software incrementally. The literature reviewed on development methods can be summarized by Benediktsson and Dalcher (2004), Benediktsson et al. (2006), and Merisalo-Rantanen et al. (2005), as the use of suitable lifecycle model (exhibiting a specific degree of flexibility) will lead to successful software projects that are delivered on time within budget and to the customer's satisfaction.

Also based on the literature review, SLOC, FP, and story point sizing methods are common methods used in the estimation process (Cohn, 2006; Galorath & Evans, 2006; Jones, 2007). These sizing methods are used in different development methodologies though story point is part of the agile development. As research in the area of sizing methods is light, especially, in the agile methodology, which has only been in existence over this last decade (Turk et al., 2005), it is hoped this study can add to the body of knowledge in this area.

Revision 11 of the ISBSG database was utilized for this research. The data from the ISBSG contains records on projects that are from different industries, applications, development lifecycles, and methods, and spans from 1989 to 2009. Of the 5,052 projects, 613 were used for this research. There were only three of the eight combinations for the 3 x 3 factorial ANOVA. Additionally, there was only one occurrence for the three valid combinations. Thus, statistical analysis for each of the hypotheses could not be performed using the proposed method.

Though the hypotheses could not be tested, the research question was answered such that there was no difference between SLOC and FP sizing methods. As there were not enough statistical samples of story point, that methods effect could not be determined. Similarly, there was also no difference in the effort accuracy error (%) between waterfall, and agile development methods while there were not enough statistical samples of incremental, however, to determine effect.

Limitations

There are three limitations that need to be recognized in this research. The first limitation was the study only collected quantitative data with no corollary subjective data that may add insight into the results. The second limitation was that it was believed projects in the database were from the better-performed part of the industry, and therefore, didn't necessarily represent failures of projects as prevalently as a random sample. Third, the ISBSG data did not have enough samples of incremental development method and story point sizing method. Though the database contained 5,052 records, the lack of complete questionnaires caused much of the data to be unusable for this study.

Recommendations

This study was among very few in the academic world to search for empirical evidence of the success factors for accurate software effort estimations. As such, it has provided a starting point for further research for related factors that impact these estimates, something that so far has been comprised of inconsistent or out-of-date evidence in the practitioner literature. Based on the limitations encountered in this study, future research needs to focus on specific factors that could affect the effort estimation accuracy.

This research has begun filling a gap in the body of knowledge and has created opportunities for future research in the area of effort estimation accuracy. This research focused primarily on sizing and development methods as factors to the effort estimation accuracy. Though this study only addressed a subset of just two factors of the estimation process, the ISBSG database contains a wealth of data to determine what other factors could contribute to estimation accuracy. As obtaining high quality data is key to future research, industry and academia should strive to provide thorough questionnaire submittals. The ISBSG database should be used, and continue to be used for future for estimation analysis. For industry, it is recommended that fully supporting these questionnaires would help in providing the data for future research.

REFERENCES

Abdel-Hamid, T. K., Sengupta, K., & Swett, C. (1999). The impact of goals on software project management: An experimental investigation. *MIS Quarterly, 23*(4), 531-555. Retrieved from ABI/INFORM Global database.

Abrahamsson, P., Moser, R., Pedrycz, W., Sillitti, A., & Succi, G. (2007). Effort prediction in iterative software development processes - Incremental versus global prediction models. In *Proceedings of the First international Symposium on Empirical Software Engineering and Measurement,* (pp. 344-353). Washington, DC: IEEE. doi:10.1109/ESEM.2007.16

Agarwal, R., Kumar, M., Yogesh, S. M., Bharadwaj, R. M., & Anantwar, D. (2001). Estimating software projects. *Software Engineering Notes, 26*(4), 60-67. doi:10.1145/505482.505491

Banker, R. D., Chang, H., & Kemerer, C. F. (1994). Evidence on economies of scale in software development. *Information and Software Technology, 36*(5), 275-282. doi:10.1016/0950-5849(94)90083-3

Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70-77. doi:10.1109/2.796139

Beck, K. (2000). *Extreme programming explained: Embrace change*. Reading, MA: Addison-Wesley.

Baheti, P. (2002). Assessing distributed pair programming. In *Companion of the 17th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 50-51). New York: Association for Computing Machinery. doi: 10.1145/985072.985099

Benediktsson, O., & Dalcher, D. (2004). New insights into effort estimation for incremental software development projects. *Project Management Journal, 35*(2), 5-12. Retrieved from ABI/INFORM Global database.

Benediktsson, O., Dalcher, D., Reed, K., & Woodman, M. (2003). COCOMO-Based effort estimation for iterative and incremental software development. *Software Quality Journal, 11*(4), 265-281. doi:10.1023/A:1025809010217

Benediktsson, O., Dalcher, D., & Thorbergsson, H. (2006). Comparison of software development life cycles: A multiproject experiment. *IEEE Proceedings Software, 153*(3), 87-101. Retrieved from IEEE Xplore database.

Berlin, S., Raz, T., Glezer, C., & Zviran, M. (2009). Comparison of estimation methods of cost and duration in IT projects. *Information and Software Technology, 51*(4), 738-748. doi:10.1016/j.infsof.2008.09.007

Boehm, B. W. (1981). *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B. W., Abts, C., Brown, A.W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., . . . Steece, B. (2000). *Software cost estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall PTR.

Boehm, B. (2006). One-size-fits-all methods: The wrong solution to new problems. Column within Ågerfalk, P. J. & Fitzgerald, B., (Eds.). (2006) Introduction. *Communications of the ACM, 49*(10), 26-34. doi:10.1145/1164394.1164416

Boehm, B. W., & Valerdi, R. (2008). Achievements and challenges in cocomo-based software resource estimation. *IEEE Software, 25*(5), 74-83. doi:10.1109/MS.2008.133

Boghossian, Z. J. (2002). An investigation into the critical success factors of software development process, time, and quality. *Dissertation Abstract International, 63*(08), 3784. (AAT 3061610).

Buglione, L., & Abran A. (2007). Improving estimations in agile projects: Issues and avenues. In *Proceedings of the 4th Software Measurement European Forum (SMEF 2007;* pp.265-274), Rome. Retrieved from http://www.dpo.it/ smef2007/papers/day2/212.pdf

Carr, M. (1997). *Software effort and schedule estimation: A case study*. (Working Paper Series), Dr Robert Davison (ed.), Department of Information, City University of Hong Kong. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi= 10.1.1.5.7643&rep=rep1&type=pdf

Cao, L. & Ramish, B. (2008). Agile requirements engineering practices: Am empirical study. *IEEE Software, 25*(1), 60-67. doi:10.1109/MS.2008.1

Choi, J., & Sircar, S. (2006, Winter). An empirical model of software size and work-effort estimation for business application software. *The Journal of Computer Information Systems*, *47*(2), 66-75. Retrieved from ABI/INFORM Global.

Cohn, M. (2004). *User stories applied: for agile software development.* Boston, MA: Addison-Wesley.

Cohn, M. (2006). *Agile estimating and planning.* Upper Saddle River, NJ: Prentice-Hall PTR.

Cooper, D. R., & Schindler, P. S. (2006). *Business research methods* (9th ed.). Boston, MA: McGraw-Hill.

Creswell, J. W. (2003). *Research design: Qualitative, quantitative, and mixed method approaches* (2nd ed.). Thousand Oaks, CA: Sage.

Creswell, J. W., & Creswell, J. D. (2005). Mixed methods research: Developments, debates, and dilemmas. In R. A. Swanson, & E. Holton, III. (Eds.), *Research in organizations: Foundations and methods of inquiry* (pp. 315-326). San Francisco, CA: Berrett-Koehler.

De Lucia, A., Pompella, E., & Stefanucci, S. (2002). Effort estimation for corrective software maintenance. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, (pp. 409-416). New York: Association for Computing Machinery. doi:10.1145/568760.568831

Devnani-Chulani, S. (1999). Bayesian analysis of software cost and quality models. *Dissertation Abstract International*, *60*(06), 2780. (UMI No. AAT 9933694)

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management, 16*(4), 88-100. Retrieved from ABI/INFORM Global.

Galorath, D. D., & Evans, M. W. (2006). *Software sizing, estimation, and risk management: When performance is measured performance improves*. Boca Raton, FL: Auerbach.

Glass, R. L. (2006). The Standish report: Does it really describe a software crisis? *Communications of the ACM, 49*(8), 15-16. doi:10.1145/1145287.1145301

Grimstad, S., & Jørgensen, M. (2006). A framework for the analysis of software cost estimation accuracy. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering* (pp. 58-65). New York: Association for Computing Machinery. doi: 10.1145/1159733.1159745

Grimstad, S., Jørgensen, M., & Moløkken-Østvold, K. (2006). Software effort estimation terminology: The tower of Babel. *Information and Software Technology, 48*(4), 302–310. doi:10.1016/j.infsof.2005.07.002

Guntamukkala, V., Wen, H. J., & Tarn, J. M. (2006). An empirical study of selecting software development life cycle models. *Human Systems Management, 25*(4), 265–278. Retrieved from Business Source Alumni Edition database.

Highsmith, J. (2001). History: The agile manifesto. Retrieved on November 18, 2008 from http://agilemanifesto.org/history.html

Hoffer, J. A., George, J. F., & Valacich, J. S. (2005). *Modern systems analysis & design* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Holmström, H., Fitzgerald, B., Ågerfalk, P. J., & Conchúir, E. Ó. (2006). Agile practices reduce distance in global software development. *Information Systems Management, 23*(3), 7-18. Retrieved from Business Source Complete database.

Holton, E. F., III. & Burnett, M. F. (2005). The basics of quantitative research. In R. A. Swanson, & E. Holton, III. (Eds.), *Research in organizations: Foundations and methods of inquiry* (pp. 29-44). San Francisco, CA: Berrett-Koehler.

International Software Benchmarking Standards Group Limited. (2008). http://www.isbsg.org/. Hawthorn Victoria, Australia.

International Software Benchmarking Standards Group Limited. (2010). *Guidelines for use of the ISBSG data.* Retrieved on 30 January 2010 from http://www.isbsg.org/isbsgnew.nsf/WebPages/10615E34A13AE0F3CA25747100422E9D/$file/Guidelines%20for%20use%20of%20the%20ISBSG%20data.pdf Author: Hawthorn Victoria, Australia.

Javed, T., Maqsood, M-E., & Durrani, Q. S. (2006). Managing geographically distributed clients throughout the project management life cycle. *Project Management Journal, 37*(5), 76-87. Retrieved from ABI/INFORM Global.

Jick, T. D. (1979). Mixing qualitative and quantitative methods: Triangulation in action. *Administrative Science Quarterly, 24*(4), 602-611. Retrieved from Business Source Complete database.

Jones, C. (2007*). Estimating software costs: Bringing realism to estimating* (2nd ed.). New York, NY: McGraw-Hill.

Jørgensen, M. (2005). Practical guidelines for expert-judgment-based software effort estimation. *IEEE Software, 22*(3), 57-63. doi:10.1109/MS.2005.73

Jørgensen, M., & Grimstad, S. (2008). Avoiding irrelevant and misleading information when estimating development effort. *IEEE Software, 25*(3), 78-83. doi:10.1109/MS.2008.57

Jørgensen, M., & Moløkken-Østvold, K. (2004). Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering, 30*(12), 993-1007. doi:10.1109/TSE.2004.103

Jørgensen, M., & Moløkken-Østvold, K. (2006). How large are software cost overruns? A review of the 1994 CHAOS report. *Information and Software Technology,* 48(4), 297-301. doi:10.1016/j.infsof.2005.07.002

Jørgensen, M., & Shepperd, M. (2007). A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering, 33*(1), 22-53. doi:10.1109/TSE.2007.256943

Keenan, F., Powell, S., Coleman, G., & McDaid, K. (2006). Learning project planning the agile way. *ACM SIGCSE Bulletin, 38*(3), 324. doi: 10.1145/1140123.1140231

Kemerer, C. (1987). An empirical validation of software cost estimation models. *Communications of the ACM, 30*(5), 416-429. doi:10.1145/22899.22906

Larman, C. (2004). *Agile & iterative development: A manager's guide.* Boston, MA: Addison-Wesley.

Lee, G., Delone, W., & Espinosa, J. A. (2006). Ambidextrous coping strategies in globally distributed software development projects. *Communications of the ACM, 49*(10), 35-40. doi: 10.1145/1164394.1164417

Lewis, J. P. (2001). Limits to software estimation. *ACM SIGSOFT Software Engineering Notes, 26*(4), 54-59. doi:10.1145/505482.505490

Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information Systems Management, 21*(3), 41-52. Retrieved from Business Source Complete database.

Liu, Q., & Mintram, R. (2006). Using industry based data sets in software engineering research. In *Proceedings of the 2006 International Workshop on Summit on Software Engineering Education* (pp. 33-36). New York: Association for Computing Machinery. doi:10.1145/1137842.1137855

MacDonell, S. G., & Gray, A. R. (2005). The viability of fuzzy logic modeling in software development effort estimation: Opinions and expectations of project managers. *International Journal of Software Engineering & Knowledge Engineering, 15*(5), 893-918. doi:10.1142/S0218194005002555

McMahon, P. E., (2006). Lessons learned using agile methods on large defense contracts. *Crosstalk, The Journal of Defense Software Engineering,* 25-30. Retrieved from http://www.stsc.hill.af.mil/crosstalk/2006/05/0605McMahon.html

Menzies, T., Chen, Z., Hihn, J., & Lum, K. (2006). Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering, 32*(11), 883-895. doi:10.1109/TSE.2006.114

Menzies, T., Port, D., Chen, Z., Hihn, J., & Stukes, S. (2005). Validation methods for calibrating software effort models. In *Proceedings of the 27th International Conference on Software Engineering*, (pp. 587-595). New York: Association for Computing Machinery. doi:10.1145/1062455.1062559

Merisalo-Rantanen, H., Tuunanen, T., & Rossi, M. (2005). Is extreme programming just old wine in new bottles: A comparison of two cases. *Journal of Database Management, 16*(4), 41-61. Retrieved from ABI/INFORM Global.

Miles, M. B. (1979). Qualitative data as an attractive nuisance: The problem of analysis. *Administrative Science Quarterly, 24*(4), 590-601. Retrieved from Business Source Complete database.

Mizuno, O., Kikuno, T., Takagi, Y., & Sakamoto, K. (2000). Characterization of risky projects based on project managers' evaluation. In *Proceedings of the 22nd International Conference on Software Engineering* (pp. 387-395). New York: Association for Computing Machinery. doi:10.1145/337180.337226

Moløkken, K., & Jørgensen, M. (2003). A review of surveys on software effort estimation. *2003 International Symposium on Empirical Software Engineering*, 223-231. Retrieved from ACM Digital Library database.

Moløkken-Østvold, K. Haugen, N. C., & Benestad, H. C. (2008). Using planning poker for combining expert estimates in software projects. *The Journal of Systems and Software, 81*(12), 2106-2117. doi:10.1016/j.jss.2008.03.058

Robiolo, G., & Orosco, R. (2007). An alternative method employing uses cases for early effort estimation. In *Proceedings of the 31st IEEE Software Engineering Workshop,* (pp. 89-98). Washington, DC: Institute of Electrical and Electronics Engineers. doi:10.1109/SEW.2007.91

Royce, W. (1998). *Software project management: A unified framework*. Reading, MA: Addison-Wesley.

Royce, W. W. (1970). Managing the development of large software systems: Concepts and techniques. In *Proceedings of IEEE WESCON,* (pp. 1-9) Washington DC:

Institute of Electrical and Electronics Engineers. Retrieved from http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Rule, G. (2000). Bees and the art of estimating. Within Boehm, B., & Fairley R. E. (2000) Software estimation perspectives. *IEEE Software, 17*(6), 23. doi:10.1109/MS.2000.895164

Rumpe, B., & Schröder, A. (2002). Quantitative survey on extreme programming projects. In *Proceedings of International Conference on Extreme Programming and Flexible Processes in Software Engineering,(XP2002;* pp. 95-100*)*. Retrieved on January 4, 2009 from https://www.agilealliance.org/system/article/file/1043/file.pdf

Ruona, W. E. A. (2005). Analyzing qualitative data. In R. A. Swanson, & E. Holton, III. (Eds.), *Research in organizations: Foundations and methods of inquiry* (pp. 233-263). San Francisco, CA: Berrett-Koehler.

Sfetsos, P., Angelis, L., & Stamelos, I. (2006). Investigating the extreme programming system - An empirical study. *Empirical Software Engineering, 11*(2), 269-301. doi:10.1007/s10664-006-6404-6

Sheetz, S. D., Henderson, D., & Wallace, L. (2009). Understanding developer and manager perceptions of function points and source lines of code. *The Journal of Systems and Software, 82*(9), 1540-1549. doi:10.1016/j.jss.2009.04.038

Succi, G., & Marchesi, M. (2001). *Extreme programming examined*. Boston, MA: Addison-Wesley.

Turk, D., France, R., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management, 16*(4), 62-87. Retrieved from ABI/INFORM Global.

USC. (2002). COCOMO. The University of Southern California, Los Angeles, CA. Retrieved on September 17, 2008 from http://sunset.usc.edu/research/COCOMOII/index.html.

Wells, G. (1999). Why projects fail. *Management Science Journal.* Retrieved on July 30, 2008 from http://www.msi.ms/MSJ/Why%20projects%20fail%20part1%209909221737.htm

Wohlin, C. (2004). Are individual differences in software development performance possible to capture using a quantitative survey? *Empirical Software Engineering, 9*(3), 211-228. doi:10.1023/B:EMSE.0000027780.08194.b0

Wu, S., & Kuan, I. (2008). A component-based approach to effort estimation. *International Journal of Soft Computing Applications, 32*(28-36). Retrieved from http://www.eurojournals.com/ijsca%203.pdf

Yang, Y., Boehm, B., & Clark, B. (2006). Assessing COTS integration risk using cost estimation inputs. In *Proceeding of the 28th International Conference on Software Engineering*, (pp. 431–438). New York: Association for Computing Machinery. doi:10.1145/1134285.1134345

APPENDIX A. FIELD DEFINITIONS

The following table shows the ISBSG database fields used in this study along with the survey question numbers related to the fields. The table consists of five rows where

- Row 1 contains the ISBSG Category
- Row 2 contains the ISBSG Field Name
- Row 3 contains the SPSS Variable Name
- Row 4 contains the Field Description
- Row 5 contains the source question numbers from questionnaire

Table A1. *ISBSG Fields Used*

Project ID	*Rating*	
	Data Quality Rating	UFP rating
Project_ID	Data_Quality_Rating	UFP_Rating
SBSG Project ID A primary key, for identifying projects in the ISBSG repository. (These Identification numbers have been 'randomized' to remove any chance of identifying a company).	Project Rating This field contains an ISBSG rating code of A, B, C or D applied to the project data by the ISBSG quality reviewers to denote the following: A = The data submitted was assessed as being sound with nothing being identified that might affect its integrity. B = The submission appears fundamentally sound but there are some factors which could affect the integrity of the submitted data. C = Due to significant data not being provided, it was not possible to assess the integrity of the submitted data. D = Due to one factor or a combination of factors, little credibility should be given to the submitted data.	Unadjusted Function Point Rating This field contains an ISBSG rating code applied to the Functional Size (Unadjusted Function Point count) data by the ISBSG quality reviewers to denote the following: A = The unadjusted FP was assessed as being sound with nothing being identified that might affect its integrity B = The unadjusted function point count appears sound, but integrity cannot be assured as a single figure was provided C = Due to unadjusted FP or count breakdown data not being provided, it was not possible to provide the unadjusted FP data D = Due to one factor or a combination of factors, little credibility should be given to the unadjusted FP data
No related question. Assigned by ISBSG	Questions 84, 85, 111, 116, 138, 139	Questions 92, 93, 97, 98 111, 116

Table A1. *ISBSG Fields Used (continued)*

Sizing		*Effort*
Count Approach	Normalised Work Effort	Summary Work Effort
Count_Approach	Normalised_Work_Effort	Summary_Work_Effort
Count Approach A description of the technique used to size the project. For most projects in the ISBSG repository this is the Functional Size Measurement Method (FSM Method) used to measure the functional size (e.g. IFPUG, MARK II, NESMA, FiSMA, COSMIC etc.). For projects using Other Size Measures (e.g. LOC etc.) the size data is in the section 'Size Other than FSM'. This helps you to compare apples with apples.	Normalised Work Effort Full life-cycle effort for all teams reported For projects covering less than a full development life-cycle, this value is an estimate of the full life-cycle effort for all reported teams. For projects covering the full development life-cycle, and projects where life-cycle coverage is not known, this value is the same as Summary Work Effort. For projects where the Summary Work Effort is not known this value is blank.	Summary Work Effort Total effort in hours recorded against the project. For projects covering less than a full development life-cycle, this value only covers effort for the phases reported. It includes effort for all reported teams. For projects covering the full development life-cycle, and projects where life-cycle coverage is not known, this value is the total effort for all reported teams. For projects where the total effort is not known this value is blank.
Question 15	Questions 68, 69, 73, 74, 75, 76	Questions 68, 69, 73, 74, 75, 76

Table A1. *ISBSG Fields Used (continued)*

Documents & Techniques	*Additional Requested Fields*		
Development Techniques	Size estimate method	Effort estimate	Planning comments
Development_Techniques	Size_estimate_method	Effort_Estimate	Planning_Comments
Development Techniques Techniques used during development. (e.g.: JAD, Data Modeling, OO Analysis etc.). Note these techniques have not been recorded as being phase specific and therefore may apply to any part of the development lifecycle.	Size estimate method Describes method used for sizing the project	Effort estimate Reported estimated project effort in hours	Planning comments Where provided, gives detailed comments on planning of the project
Question 6	Question 15	Question 16	Question 21

Derived and Calculated Field for this Study			
Development Method	Sizing Method	Normalization Ratio	Effort Accuracy Error (%)
Development_Method	Sizing_Method	Normalization_Ratio	Accuracy
Development Method Added field for this study providing enumerated type from Development Techniques field 1=Waterfall 2=Incremental 3=Agile	Sizing Method Added field for this study providing enumerated type from Size estimate method field 1=SLOC 2=Function Point 3=Story Point	Normalization Ratio Added field for this study providing the calculated normalization ratio from the Normalised Work Effort and the Summary Work Effort	Effort Accuracy Error (%) Added field for this study providing the calculated Effort accuracy Error from the Effort Estimate and Normalized Work Effort
N/A	N/A	N/A	N/A

APPENDIX B. SPSS SYNTAX CODE

```
/* Calculates the Variable Accuracy from the Normalized Work Effort and
Effort Estimate */

COMPUTE Accuracy = ((Normalised_Work_Effort  - Effort_Estimate )/
  Effort_Estimate) *100.
  EXECUTE .

/* Calculates the Variable Normalization Ratio from the Normalised
[sic] Work Effort and Summary Work Effort */

COMPUTE Normalization_Ratio = Normalised_Work_Effort /
  Summary_Work_Effort.
  EXECUTE .

/*Provides a series of boxplots of accuracies by Sizing Method */

EXAMINE
VARIABLES=Accuracy BY Sizing_Method
  /PLOT=BOXPLOT/STATISTICS=NONE/NOTOTAL
  /MISSING=REPORT.

/*Provides a series of boxplots of accuracies by Development Method */

EXAMINE
VARIABLES=Accuracy BY Development_Method
  /PLOT=BOXPLOT/STATISTICS=NONE/NOTOTAL
  /MISSING=REPORT.

/*Provides three sets of frequency tables and barcharts for Development
Method Sizing Method and Accuracy */

FREQUENCIES
  VARIABLES=Development_Method Sizing_Method Accuracy
  /NTILES=  4
  /STATISTICS=MEAN VARIANCE MINIMUM MAXIMUM
  /BARCHART  FREQ
  /ORDER=  ANALYSIS .

/*Provides cross tab for Development Method by Sizing Method */

CROSSTABS
  /TABLES=Development_Method BY Sizing_Method
  /FORMAT= AVALUE TABLES
  /STATISTIC=CHISQ CORR
  /CELLS= COUNT
  /COUNT ROUND CELL .

/* Provides an ANOVA exploring the effects of Sizing Method on Accuracy
*/

UNIANOVA
  Accuracy  BY Sizing_Method
  /METHOD = SSTYPE(3)
  /INTERCEPT = INCLUDE
  /PLOT = PROFILE( Sizing_Method)
  /EMMEANS = TABLES(Sizing_Method)
  /PRINT = DESCRIPTIVE ETASQ
  /CRITERIA = ALPHA(.05)
  /DESIGN = Sizing_Method .

/* Provides an ANOVA exploring the effects of Development Method on
Accuracy */

UNIANOVA
  Accuracy  BY Development_Method
  /METHOD = SSTYPE(3)
  /INTERCEPT = INCLUDE
  /PLOT = PROFILE( Development_Method )
  /EMMEANS = TABLES(Development_Method)
  /PRINT = DESCRIPTIVE ETASQ
  /CRITERIA = ALPHA(.05)
  /DESIGN = Development_Method .


/* The following provides the 3 x 3 factorial analysis for the five
hypotheses in the study.  Initial code was provided by Dr. Garvey
House, Capella University */

/* Provides a Generalized Linear Models- Test of Factorial ANOVA with
marginal mean descriptive statistics */

GENLIN Accuracy BY Development_Method Sizing_Method (ORDER=ASCENDING)
   /MODEL Development_Method Sizing_Method
Development_Method*Sizing_Method INTERCEPT=YES
DISTRIBUTION=NORMAL LINK=IDENTITY
  /CRITERIA METHOD=FISHER(1) SCALE=MLE COVB=MODEL MAXITERATIONS=100
MAXSTEPHALVING=5
    PCONVERGE=1E-006(ABSOLUTE) SINGULAR=1E-012 ANALYSISTYPE=3(WALD)
CILEVEL=95 CITYPE=WALD
    LIKELIHOOD=FULL
  /EMMEANS SCALE=ORIGINAL
  /EMMEANS TABLES=Development_Method SCALE=ORIGINAL
COMPARE=Development_Method CONTRAST=DEVIATION PADJUST=LSD
  /EMMEANS TABLES=Sizing_Method SCALE=ORIGINAL PADJUST=LSD
  /EMMEANS TABLES=Development_Method*Sizing_Method SCALE=ORIGINAL
PADJUST=LSD
  /MISSING CLASSMISSING=INCLUDE
  /PRINT CPS DESCRIPTIVES MODELINFO FIT SUMMARY SOLUTION LMATRIX
HISTORY(1) .
```

```
/* Provides a Generalized Linear Models –Test of simple effect marginal
means for sizing method */

GENLIN Accuracy BY Development_Method Sizing_Method (ORDER=ASCENDING)
  /MODEL Development_Method(Sizing_Method) INTERCEPT=YES
DISTRIBUTION=NORMAL LINK=IDENTITY
  /CRITERIA METHOD=FISHER(1) SCALE=MLE COVB=MODEL MAXITERATIONS=100
MAXSTEPHALVING=5
    PCONVERGE=1E-006(ABSOLUTE) SINGULAR=1E-012 ANALYSISTYPE=3(WALD)
CILEVEL=95 CITYPE=WALD
    LIKELIHOOD=FULL
  /EMMEANS SCALE=ORIGINAL
  /EMMEANS TABLES=Development_Method SCALE=ORIGINAL
COMPARE=Development_Method CONTRAST=DEVIATION PADJUST=LSD
  /EMMEANS TABLES=Sizing_Method SCALE=ORIGINAL PADJUST=LSD
  /EMMEANS TABLES=Development_Method*Sizing_Method SCALE=ORIGINAL
PADJUST=LSD
  /MISSING CLASSMISSING=INCLUDE
  /PRINT CPS DESCRIPTIVES MODELINFO FIT SUMMARY SOLUTION LMATRIX
HISTORY(1) .

/* Provides a Generalized Linear Model - Test of simple effect marginal
means for development method */

GENLIN Accuracy BY Development_Method Sizing_Method (ORDER=ASCENDING)
  /MODEL Sizing_Method(Development_Method) INTERCEPT=YES
DISTRIBUTION=NORMAL LINK=IDENTITY
  /CRITERIA METHOD=FISHER(1) SCALE=MLE COVB=MODEL MAXITERATIONS=100
MAXSTEPHALVING=5
    PCONVERGE=1E-006(ABSOLUTE) SINGULAR=1E-012 ANALYSISTYPE=3(WALD)
CILEVEL=95 CITYPE=WALD
    LIKELIHOOD=FULL
  /EMMEANS SCALE=ORIGINAL
  /EMMEANS TABLES=Development_Method SCALE=ORIGINAL
COMPARE=Development_Method CONTRAST=DEVIATION PADJUST=LSD
  /EMMEANS TABLES=Sizing_Method SCALE=ORIGINAL PADJUST=LSD
  /EMMEANS TABLES=Development_Method*Sizing_Method SCALE=ORIGINAL
PADJUST=LSD
  /MISSING CLASSMISSING=INCLUDE
  /PRINT CPS DESCRIPTIVES MODELINFO FIT SUMMARY SOLUTION LMATRIX
HISTORY(1).

/* Provides univariate ANOVA with Profile Plots and descriptive
statistics */

UNIANOVA Accuracy BY Development_Method Sizing_Method
  /CONTRAST(Development_Method)=Deviation
  /METHOD=SSTYPE(3)
  /INTERCEPT=INCLUDE
  /PLOT=PROFILE(Development_Method*Sizing_Method
Sizing_Method*Development_Method)
  /PRINT=ETASQ HOMOGENEITY PARAMETER DESCRIPTIVE OPOWER GEF LOF
TEST(LMATRIX)
  /PLOT=SPREADLEVEL RESIDUALS
  /CRITERIA=ALPHA(.05)
  /DESIGN=Development_Method Sizing_Method
Development_Method*Sizing_Method .
```