2015

Exploring Government Contractor Experiences Assessing and Reporting Software Development Status

Putnam P. Texel
Walden University

Walden University

College of Management and Technology

This is to certify that the doctoral dissertation by

Putnam Texel

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

Review Committee
Dr. Branford McAllister, Committee Chairperson, Management Faculty
Dr. David Gould, Committee Member, Management Faculty
Dr. Robert Kilmer, University Reviewer, Management Faculty

Chief Academic Officer
Eric Riedel, Ph.D.

Walden University
2015

Abstract

Exploring Government Contractor Experiences Assessing and Reporting Software Development Status

by

Putnam P. Texel

MS, Fairleigh Dickinson University, 1968

BA, Fairleigh Dickinson University, 1967

Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Management

Walden University

February 2015

Abstract

Reports from academic, commercial, and government organizations have documented software-intensive system cost and schedule overruns for decades. These reports have identified lack of management insight into the software development process as one of many contributing factors. Multiple management mechanisms exist. However, these mechanisms do not support the assessment, and subsequent reporting, of software completion status. Additionally, the conceptual framework, based on industry standards, is limited in its relevance to this study due to an emphasis on what is needed while deferring implementation details. The purpose of this phenomenological study was to explore U.S. government contractors' lived experiences of assessing and reporting software completion status with current measurement mechanisms. Twenty program or project managers responded to interview questions targeting positive and challenging experiences with current measurement mechanisms. Qualitative analysis of the experiential data was based on open and axial coding conducted on interview transcripts. Analysis indicated that costly resources are applied to metrics that do not provide the required level of management insight into completion status. These findings have positive social change implications for program managers, project managers, and researchers by documenting the need to develop relevant and cost-efficient status metrics to provide the critical insight required by management to reduce overruns.

Exploring Government Contractor Experiences Assessing and Reporting Software Development Status

by

Putnam P. Texel

MS, Fairleigh Dickinson University, 1968

BA, Fairleigh Dickinson University, 1967

Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Management

Walden University

February 2015

Dedication

This dissertation is dedicated to my grandfather, Ruel Putnam Pope, who made this journey possible; my grandmother, Ruth Patillo Griffin Pope, who taught me grace and humility; and my grandchildren, Myles, Morgan, and Finn Texel, all of whom I hope will grow to treasure the continual gift of learning.

Acknowledgments

The following individuals created a team composed of different strengths and skills that, when taken as a whole, provided an A++ support system.

- My sister, Penny Prichard, my soul mate, for her encouragement, support, inspiration, and, above all, never failing to answer the telephone and listen.
- My friends and colleagues, Trisha-Marie Uhl, Linda May, Betsey Nohe, Cheryl Douthitt, Lori Larson, and Colleen Hanrahan, for being who they are—strong, intelligent women who never forget caring and compassion.
- Dr. Mark Ellinger and Dr. William Tollefson, who supported me through this journey and were instrumental in the achievement of my goal.

And last, but by no means least, to my chair, Dr. Branford McAllister, whose never-failing optimism, encouragement, and focused input kept me continually striving to improve; to Dr. David Gould, for his encouragement and discussions that proved so beneficial; to Dr. Robert Kilmer, for his time and expertise in this problem domain as well as his knowledge of APA mechanics; to John Tripp of Academic Advising, for his help and knowledge of the PhD process; to the Writing Center and Library staff for their experience and professionalism; and to Student Technical Support for their help as we transitioned to Blackboard and Google mail.

All these individuals contributed to the fulfillment of my dream, and I am deeply grateful to each and every one.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction to the Study

Cost and schedule overruns associated with software-intensive systems have been well documented, with lack of management insight into the software development process repeatedly identified as a contributing factor to both overruns and failure (National Defense Industrial Association [NDIA], 2010; The Standish Group, 2010; U.S. Department of Defense Government Accountability Office [USDOD], 2011). In this chapter, I begin by identifying a gap within a continuum of existing software development measurement mechanisms and describing this gap in an introductory literature review. Subsequent subsections contain descriptions of the problem associated with that gap, the purpose of the research effort, the research questions, assumptions, limitations, and delimitations of the study.

Background of the Study

Software-intensive systems are the backbone of multiple industries, including but not limited to defense, education, finance, health, and transportation. A software-intensive system, as defined in ISO/IEC/IEEE Standard 42010, is "any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole" (Joint Technical Committee JTC-1 Subcommittee 7 [JTC1SC7], 2011, p. 4). Software-intensive systems are characterized by complexity, multiple and diverse requirements, and multiple and diverse stakeholders.

Despite many successful deployments of software-intensive systems, both commercial and government research reports have documented a history of cost and schedule overruns. Overruns have plagued the software community for decades. The

Standish Group is a commercial organization dedicated to researching software project failures to improve the probability of success. Starting in 1995, researchers from the Standish Group conducted a survey of 365 participants from small-, medium-, and large-scale companies to analyze success and failure rates of software projects (Standish Group, 1995). Beginning with this 1995 study is a conscious choice because, as will be seen in this chapter as well as in Chapter 2, there has been no significant change in success and failure rates from 1995 to the present day.

The companies in The Standish Group's 1995 CHAOS Report represented multiple vertical markets (e.g., finance, retail) and 8,380 applications (Standish Group, 1995). Follow-up focus groups and personal interviews provided qualitative support for the study. Analysis of the response data indicated that 31.1% of the contracts would be cancelled and 52.7% of the projects would have an average cost overrun of 189%. Additionally, the results forecasted that $81 billion would be lost on cancelled software projects and an additional $59 billion would be attributed to cost overruns. A valid concern is whether the response data were properly cleaned prior to analysis, specifically whether multiple respondents specified the same application. The last two paragraphs of this subsection address additional concerns with respect to research conducted by the Standish Group.

Continuing into the 21st century, research conducted by the Standish Group (2004) indicated that in the United States alone, 71% of software projects encountered cost and schedule overruns, and total waste was estimated at $55 billion per year. A 2009 Government Accountability Office (GAO) Report, GAO-09-326SP, identified a $6.9-

billion overrun sustained by 10-enterprise resource planning (ERP) systems (USDOD, 2009). A subsequent 2010 GAO report, GAO-10-1059T, identified a $296 billion overrun, with 64 out of 96 weapons programs exceeding initial cost estimates (USDOD, 2010).

As illustrated in Table 1, the Standish Group frequently updates research findings to provide data relevant to the annual percentage of projects categorized as successful, challenged, or impaired. Although improvement in successful projects increased slightly, the findings in Table 1 are not without controversy. Challenging the findings of the Standish Group, Eveleens and Verhoef (2010) questioned the Standish Group categorization of successful, challenged, and failed projects and put forth that the definitions of the categories did not include (a) the possibility of meeting cost and schedule with reduced functionality and (b) program underruns.

A subsequent study by the Center for Strategic and International Studies (CSIS), a nonprofit research organization dedicated to providing insight for decision makers, concluded that Standish Group findings did not account for the fact that newer program data may not accurately represent project data because program data change as programs age (Hofbauer, Sanders, Ellman, & Morrow, 2011). Lastly, Jørgensen and Molokken (2006) conducted additional research on the figures presented by the CHAOS Group and found a lack of transparency in the methodology used. However, despite critiques of Standish Group research, the Standish Group remains a primary source of research in the software industry today and is repeatedly referenced.

Table 1

Standish Report Research Figures 1994–2009

Standish Group summary findings			
Year	Successful	Challenged	Failed
1994	16%	53%	31%
2004	29%	53%	18%
2009	32%	44%	24%
2013	39%	43%	18%

Note. Data extracted from "The Rise and Fall of the CHAOS Report Figures," by J. L. Eveleens and C. Verhoef, 2010, *IEEE Software, 27*(1), 30-36, and *The CHAOS Manifesto 2013*, by The Standish Group, 2013, retrieved from http://www.versionone.com/assets/img/files /CHAOSMnifesto2013.pdf

Problem Statement

For decades, many obstacles have been identified that impede the development and deployment of software-intensive systems within cost and schedule constraints (DeMarco, 1995a, 1995b, 1995c, 1997; Jones, 2010; NDIA, 2010; Yourdon, 2004). Requirements volatility (Dev & Awathi, 2012; Hull, Jackson, & Dick, 2011), inadequate estimation techniques (Singh, Singh, & Singh, 2011), and ineffective management (NDIA, 2010; Whitfield, 2007) are examples of repeatedly cited obstacles. My research focused on the latter, ineffective management.

With respect to ineffective management, there is a gap in current research and corresponding literature within the continuum of management mechanisms that are relevant to software development completion status. A *continuum* is a sequence of smaller elements that progress in steps to create a complete whole element. The two end points of the continuum are (a) abstractions of management truisms, which are abstract

guidelines based on experiences (Brooks, 1995; DeMarco, 1982; Yourdon, 2004), and (b) detailed syntactic metrics (Abran, 2010; Stein, 2004). The problem is that between the two ends of the continuum, there is little research on mechanisms that support software managers in their effort to successfully capture and report incremental software development progress. A bridge does not exist to connect the two distinct approaches to software measurement. A continuum of measurement does not exist. All that exists are the two ends, which approach software status measurement from two completely different levels of abstraction. Consequently, managers must rely on a combination of abstract truisms and detailed syntactic metrics to assess and report software completion status. Unfortunately, neither truisms nor syntactic measures target completion status.

Additionally, a second gap exists between academic and practical approaches to the issue (Abran, 2010; Day, 2009). As a result, those in management are often placed in a compromising position of having to explain progress without adequate backup status data.

Purpose of the Study

The purpose of this phenomenological study was to explore and describe government contractor experiences with existing software metrics. The study focused on how government contractor program and project managers view the relevance of software measurement mechanisms based on both their positive and negative experiences reporting and assessing software development completion status. Discovering their current lens with respect to the existing software management mechanisms and their alignment, or lack thereof, with software completion may confirm, or not, the existing

gap in the literature. Without an understanding of program and project managers' experiences, the software industry cannot move forward and address a second gap in the literature, that between practical and academic approaches to software metrics (Abran, 2010; Day, 2009).

Similar to the gap in management mechanisms, academic and heuristic approaches to software metrics vary (Abran, 2010; Day, 2009). Abran (2010) stated that research does not trickle down to practitioners. Contractors need a voice in the community to explain their experiences with software metrics and the results of those experiences with respect to reporting and assessing software development completion status. Cost and schedule overruns must be addressed from multiple perspectives to improve the development and deployment of successful software-intense systems. One perspective, specifically the perspective of status measurement mechanisms, was the focus of this research.

Researchers continue to identify more and more metrics, with papers on component-based software system (CBSS) metrics (Abdellatief, Sultan, Ghani, & Jabar, 2013) and agile software development metrics (Aktunc, 2012; Farid & Mitropoulos, 2013; Misra & Omorodion, 2011; Tabib, 2013; Tarhan, 2014). However, metrics that address software development completion status are missing in the literature. For example, one sample management metric applicable to an agile software development approach involves the evaluation of team domain expertise, team previous experience working together, and the inclusion of new technology, which, when combined, indicate project risk (Farid & Mitropoulos, 2013).

Research Questions

For my phenomenological research study, I focused on the following open-ended central research question: *What meaning do government contractors ascribe to their experiences with software metrics relevant to assessing and reporting software completion status?* Typical of a qualitative study, the central, or key, research question was exploratory in nature and was supported by more specific research questions (Creswell, 2007). Table 2 lists the three supporting research questions that supported the central research question.

Table 2

Research Questions

	Research questions
RQ1	How have current software metrics supported the assessment of software development completion status as perceived by program and project managers?
RQ2	How have current software metrics supported the reporting of software development completion status as perceived by program and project managers?
RQ3	What is the relevancy of software metrics to Software Development Life Cycle (SDLC) phases?

Conceptual Framework

In an effort to reduce cost overruns, members of the National Defense Industrial Association (NDIA), an association focused on ensuring information exchange between the government and defense industry, convened a workshop in 2010 to address the major obstacles associated with successful development and deployment of software-intensive systems (NDIA, 2010). Participants from industry, academia, and government focused

on identifying progress toward improvement in areas that had previously been identified as obstacles in a 2006 workshop (NDIA, 2010).

Two out of nine obstacles identified in the 2006 workshop were relevant to this research:

- Software development life cycles (SDLCs) suffered from lack of planning and management.
- Traditional management techniques do not scale to new technologies and the increased complexity of software systems (NDIA, 2010).

The 2010 workshop findings with respect to progress in addressing these two obstacles were the following:

- There was no progress in the planning and management of a SDLC.
- There was increased focus, but no real improvement, with respect to verification of techniques to address the increased complexity and rapid pace of changing technology for software systems (NDIA, 2010).

With respect to the second item, the NDIA members found that existing measures had not evolved in concert with software development technologies (NDIA, 2010). This lag in the evolution of measures resulted in increased lack of insight into software development progress. The members stated the need for future research focused on identifying measures applicable to complex software systems developed with new technologies. One contradictory result surfaced, specifically a need to improve management's effectiveness (the first obstacle) as well as an indication that the measures and indicators were not available to support that mission (the second obstacle).

In addition to lack of status data and continual emergence of new technology, my research indicated that existing frameworks provided by recognized professional organizations are of little value to software measurement needs. The Institute of Electronic and Electrical Engineers (IEEE), a global professional organization for engineers, published 367 standards relevant to software and systems engineering (IEEE Standards Association [IEEE SA], 2013). Of those 367 standards, 128 remain active, with the remaining 239 standards either superseded or withdrawn. Of the 128 active standards, 51 focus on some aspect of software. Table 3 contains a list of the 51 active software standards, categorized by focus.

Only four of the 51 active software-related IEEE standards, identified in Table 4, had even minimal relevance to my study, and they date back to 2006. Items 3 and 4 in Table 4 were not of value to this study due to focus on a specific measure, specifically dependability and quality, respectively. Item 1, vocabulary, provides 3,349 definitions of basic terms used in the systems and software engineering communities. Included are definitions for measure, measurand, metric, and indicator, which contributed to the object-oriented concept model offered by Texel (2013) in an effort to highlight the need for consistency in the definitions of these terms. Item 2 focuses solely on processes, products, and the necessity for status but lacks specifics for capturing status.

The existing framework of active IEEE standards and the frameworks within each of the 51 standards related to software were limited in their ability to support this study. Only one standard, IEEE Std 1061™-1998(R2009), focused on measurement and, like the other standards, was of limited value due to the (a) focus on quality not completion

Table 3

Categorization of Active IEEE Software Standards

Categorization of 51 active IEEE software standards	
Anomalies	1
Acquisition	1
Computer aided software engineering (CASE) tools	3
Configuration management (CM)	1
Design & architecture	2
Documentation	8
Management	3
Measurement	1
Modeling & languages	4
Process	9
Quality, assurance, & dependability	6
Reliability	1
Reuse	1
Reviews & audits	1
Risk	1
Safety	1
Testing	3
Verification & validation	1
Vocabulary	1
Website	2
Total	51

Table 4

Relevant IEEE Software Standards

IEEE standards relevant to conceptual framework				
	Standard ID	Title	Emphasis	Notes
1	ISO/IEC/IEEE 24765 (JTC1SSESC, 2007)	Systems and Software Engineering—Vocabulary	Definitions of software engineering terms.	Measure is a variable to which a value is assigned. A measure for software completion status is absent.
2	ISO/IEC TR 15939 (JTC1SC7, 2010)	Systems and Software Measurement Process	Emphasis is on measuring software development processes and products.	Measure is a variable to which a value is assigned. A measure for software completion status is absent.
3	IEEE Std 1061™-1998(R2009) (SESC, 2009)	IEEE Standard Dictionary of Measures of the Software Aspects of Dependability	Emphasis is on looking at dependability in terms of faults/lines of code, or fault density.	Dependability (defect density). Completion status is absent.
4	IEEE Std 982.1™-2005 (SESC, 2006)	IEEE Standard for a Software Quality Metrics Methodology	Emphasis is on the aspects of quality.	Dependability (defect density). Completion status is absent.

status, (b) emphasis on process rather than the identification of specific metrics, and (c) identification of the need for metrics while leaving the identification of specific metrics to individual organizations (Software Engineering Standards Committee [SESC], 2005). IEEE standards represent national standards that were of limited value to my study. International standards were equally insufficient.

One international standard, ISO/IEC TR 19759:2005, also known as *Software Engineering—Guide to the Software Engineering Book of Knowledge* (SWEBOK), represents a holistic approach to the identification and documentation of the components of the software engineering discipline (JTC1SC7, 2011). The SWEBOK provides a

framework for the development of an education and training curriculum that, in turn, could be used as a basis for software engineer certification. Major components of the SWEBOK are the (a) life cycle phases of software development, (b) management, (c) processes, and (d) quality. These four components are addressed at a level of abstraction that indicates what must be considered but not how to implement them. With respect to completion status, the SWEBOK specified that support is needed to address "implementation status, and verify compliance with specified requirements" (p. 7-1). The SWEBOK did not contribute to my study.

Lastly, the Carnegie Melon University (CMU) Software Engineering Institute's (SEI) Capability Maturity Model Integrated (CMMI) focuses on what needs to occur with respect to measures but not how, leaving customization and implementation details to an individual organization (CMUSEI, 1995). The CMMI was minimally relevant to this study, the relevance being its role in an attempt to curb overruns at the request of the Department of Defense. The content of the CMU SEI CMMI remains unchanged today, and the relevance, although minimal, of the CMMI to this study is further explored in Chapter 2.

To summarize, existing frameworks were limited in their relevance to this study. The framework of a continuum of existing management monitoring mechanisms supports the existence of the gap in the literature and highlights the need to identify measures that bridge the gap between the two ends of the continuum—specifically, the need to address software completion status measures.

A Continuum of Mechanisms to Monitor Completion Status

As previously stated, the current mechanisms that support management of software systems exist on a continuum. As illustrated in Figure 1, on one end of the continuum are abstractions of management truisms that have stood the test of time (Brooks, 1995; DeMarco, 1982; Yourdon, 2004). At the opposite end of the continuum are detailed syntactic metrics (Abran, 2010; Abreu, 1995; Chidamber & Kemerer, 1994; Lorenz & Kidd, 1994; Stein, 2004). The gap between the two ends is very wide.

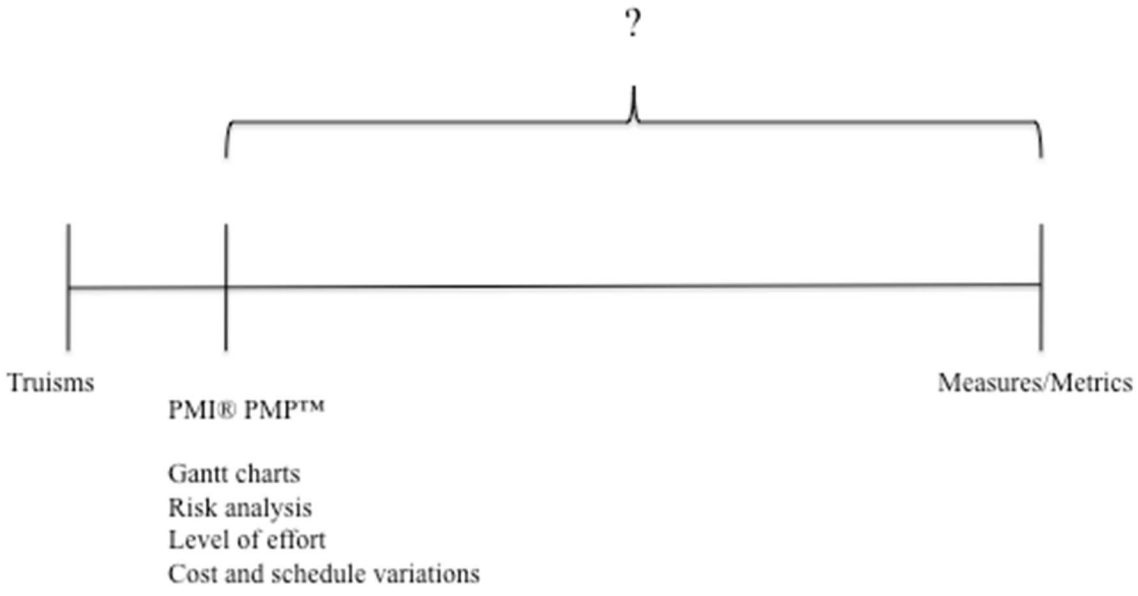


Figure 1. Continuum of management mechanisms (figure created by P. Texel using Microsoft PowerPoint).

Subsequent subsections contain examples of truisms and syntactic metrics. But first, as depicted in Figure 1, an acknowledgment of the Program Management Institute (PMI®) is necessary. In the continuum, the PMI® provides a Program Management Professional (PMP™) certification exam to ensure that managers possess basic

management skills. The PMP™ exam addresses standard management mechanisms such as scheduling, work breakdown structure (WBS), risk analysis, effort, and estimation. However, the exam does not address management of software systems, nor is that the stated intent (Mulcahy, 2011).

Abstract truisms. A classic example of a management truism initially addressed by Brooks in 1975, and then restated by Brooks in 1995, indicates,

> Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—no inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large scale integration did for computer hardware. We cannot expect ever to see two-fold gains every two years. (p. 181)

The concept of a silver bullet, a source of discussion for over 40 years, is still a topic of discussion today. In 2007, the organizing committee of the 22nd International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) convened a panel. The panel, composed of seminal software theorists and practitioners, addressed the current thinking on Brooks's position that a silver bullet did not exist to cure the ills of the software engineering discipline. The panel's conclusion remained the same: There is no silver bullet (Fraser & Manci, 2008). However, Blaha (2004) stated that smaller steps, termed *copper bullets*, led to improved software quality. Berry (2008) proposed lighter bullets, labeled *aluminum bullets*, to address the needs of the software engineering community.

A second classic truism, Brooks's law, indicates that adding staff to a failing project has negative consequences (Brooks, 1995). Multiple seminal authors in the field of software engineering offered additional truisms ranging from the importance of managing risk (DeMarco & Lister, 2003) to how to make the most out of death march projects (Yourdon, 2004). Although many truisms are still valid today, they do not provide the requisite data for monitoring completion status. Truisms are further explored in Chapter 2.

Detailed syntactic metrics. On the opposite end of the continuum, the majority of existing software development metrics for a SDLC are either based on counts, or arithmetic calculations based on counts, that are extracted from program code. These metrics are called *syntactic metrics* because they are extracted from program code based on the grammar, or syntax, of the program code. Two examples of commonly used syntactic metrics are the number of source lines of code (SLOC; i.e., how many lines of program code exist), and the cyclomatic complexity measure (CCM; i.e., algorithmic complexity).

There are multiple issues related to syntactic metrics. The first issue is the scope of existing syntactic metrics, specifically the applicability to (a) quality, (b) complexity, (c) size, and (d) level of effort, not completion status (see Appendix A). Secondly, there are too many metrics, with new metrics continually added (Abran, 2010; Aktunc, 2012; Concas, Marchesi, Murgia, & Tonelli, 2010; Farid & Mitropoulos, 2013; Gandhi & Bhatia, 2012; Iqbal & Khan, 2012; Tabib, 2013). In the presence of too many metrics, (a) metrics are often ignored and not used to guide a project (Bouwers, Visser & Van

Deursen, 2012), and (b) the Hrair limit is violated, leading to cognitive confusion (Miller, 1956). The ambiguity in the literature with respect to the interpretation of commonly used metrics is a third major issue with syntactic metrics. These issues, and more, surrounding existing software syntactic measures and metrics are further explored in Chapter 2.

As stated by one seminal author, "You can't control what you can't measure" (DeMarco, 1982, p. 3). PMP™ certification, Gantt charts, risk analysis, level of effort, and cost and schedule variations are valuable management mechanisms that are the foundation, and de facto standard, for management processes today. These latter mechanisms are (a) embedded in the management of software development efforts, (b) thoroughly addressed in the literature, and (c) not the focus of this dissertation.

Nature of the Study

A qualitative phenomenological research effort, centered on semistructured interviews, was an appropriate choice to explore government contractor program and project managers' experiences with respect to the relevance of currently available management metrics for reporting and assessing software completion status. The accessible population consisted of members of four prominent government contractors. These four contractors develop software-intensive systems for both Department of Defense (DOD) and non-DOD agencies. The contractors provided a sampling frame based on filter criteria (e.g., years of experience, availability). The sampling frame led to sample selection. The sample size, 20, reflected consideration of the breadth and depth of the research requirements. The research process protected the privacy of participants

as specified by the U.S. Department of Health and Human Services (HHS) Office of Human Research Protections (OHRP) and Walden University's Institutional Review Board (IRB).

Data Capture

Two data-capture mechanisms were employed for this study: a questionnaire and an interview protocol. Questionnaires captured participants' demographic data, and an interview protocol, consisting of semistructured interview questions, supported the capture of participants' experiences (Janesick, 2011; Patton, 2002). NoNotes.com, a third party service, supported the interview process by recording and transcribing each interview into a text file that was used for member checking and subsequently imported to NVivo for analysis.

I conducted a pilot study to support the development and test of the questionnaire and interview protocol. The sole purpose of the pilot study was to support content validity and ensure that the questionnaire and interview questions elicited the requisite data, specifically participants' satisfaction of selection criteria and experiences with software metrics, and their relevance to the assessment and reporting of software completion status.

Data Analysis

I used two software packages to support data analysis. IBM SPSS v21.0, a quantitative analysis tool, supported the generation of descriptive statistics on the demographic data captured in the questionnaires, and NVivo v10, a qualitative analysis tool, supported the analysis of the researcher-coded qualitative data captured from the

semistructured interviews (Bazeley & Jackson, 2013). The nature of the study, including data capture and data analysis, is further refined in Chapter 3.

Definitions

Attribute: A characteristic of a class (Alhir, 1998). For example, size is a characteristic, or attribute, of software; room number is a characteristic, or attribute, of room.

Class: An abstraction of a real-world entity implemented as a Java or C++ class or an Ada package. Classes for a hotel reservation system include, but are not limited to, room, guest, and reservation (Alhir, 1998).

Completion status: The current status of the implementation of stated and agreed-to software requirements with completion defined as coded, tested, and accepted by the client. (This definition generated by P. Texel for this study.)

Component: An assembly of software modules (JTC1SC7, 2010). A logically related collection of software functionality (e.g., subsystem, category, or Java package) or a hierarchy of classes based on aggregation or inheritance.

Hrair limit: The number of concepts that can be cognitively retained in memory at one time, specifically, 7 ± 2 (Miller, 1956).

Indicator: A metric targeting a characteristic, or attribute, of software when compared to a baseline (Texel, 2013). Example: combination of SLOC with comments t that together form the metric comments/SLOC that can be compared against a predetermined project baseline and form an indicator of documentation coverage.

Insight: The ability to gage the true nature of a process or entity; an "ah ha" moment (DeMarco, 1995a, p. ix).

Measure: A single quantitative value, a number, applicable to an attribute, or characteristic, of a real-world entity (Abran, 2010).

Metric: A combination of two or more measures providing the context lacking in a measure that, when combined with a baseline, forms an indicator (Abran, 2010).

NOM: An object-oriented metric representing the number of methods in a class. How to actually obtain the count differs among researchers (Chidamber & Kemerer, 1995; Churcher & Sheppard, 1995).

Project manager: An individual responsible for the software component of a project or responsible for the project (Mulcahy, 2011).

Program manager: An individual responsible for multiple projects (Mulcahy, 2011).

SLOC: Source lines of code. A single definition is not possible due to multiple approaches to counting SLOC that yield different results. Refer to Chapter 2 for more detail on issues related to SLOC.

Syntactic measure/metric: A measure or metric that is extracted from program code (Stein, 2004; Stein et al., 2009).

Assumptions

Because of signed confidentiality agreements, the primary assumption was that participants would be forthcoming with their responses to interview questions. I encouraged candid reflection on experiences in my opening statement because

conventional corporate responses would not have provided the necessary data. Additionally, adherence to ethics guidelines supporting participants' right to privacy supported and encouraged participants' candid responses.

A second assumption was that organizations would not interfere with a staff members' participation. Stated differently, upper management would not instruct participants in how to respond. With budget cuts and heavy workloads, a third assumption was that participants would be able to allocate uninterrupted time to the study. The final assumption was that each participant would be able to find a quiet, private, enclosed area in which to participate by telephone.

Scope and Delimitations

This study focused on contractors who had developed, or who were currently developing, software-intensive applications for a government agency, DOD or non-DOD. Program managers and software project managers from contractor organizations represented the target population. My research effort focused on exploring their experiences in assessing and reporting software completion status with existing monitoring mechanisms.

The intersection of two sets, personnel and software development artifacts, represents the scope of this effort. Each of these two sets could have been further decomposed into subsets. However, a complete decomposition of sets into subsets did not serve the intended purpose of this effort, nor was a complete decomposition required for this effort. What was required was to define the specific subsets that represent the scope of this effort.

Figure 2 depicts the intersection of the specific subsets of the two major sets as follows: a subset of personnel—program and project managers—and a subset of software artifacts—software metrics and completion status. Specifically excluded from the study were system engineers, software and hardware engineers, test staff, administrative personnel, configuration management staff, and technical editors.

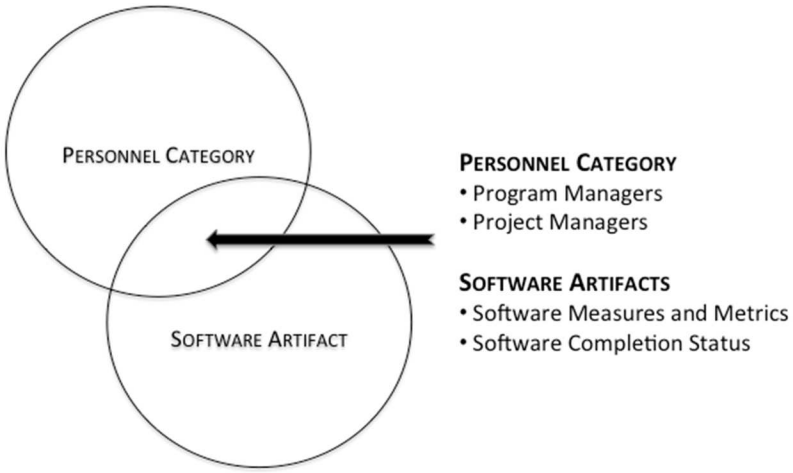


Figure 2. Research scope (figure created by P. Texel using Microsoft PowerPoint).

Limitations

There were five limitations to this research. The first limitation, supported by Patton (2002), was that sample size in qualitative analysis is not based on rules or precise mathematical analysis. To ensure meaningful results, the researcher must gauge the breadth and depth required to achieve the richness and robustness of the data collected. Too many participants may result in sacrificing depth of data, and too few participants

may result in sacrificing breadth of data. There were 20 participants for this study who represented four organizations that contract to the U.S. government.

The second limitation was researcher bias. With over 25 years of experience saving floundering or failing object-oriented (OO) software projects, I have my perspective on the industry and on managing software projects. The third limitation was the sample itself. The sampling frame was limited to those personnel who were available to participate. There was a possibility that lack of input from those who were too busy to participate might impact the results. Additionally, the sample was limited to four contractor organizations.

The fourth and fifth limitations focus on transferability and dependability. Shenton (2004) defined *transferability* in qualitative research as the ability for a reader to transfer the content of a study to the reader's experience. Sufficient detail must be present in the study for a reader to compare the study to the reader's context. I have provided this requisite level of detail in the collection and analysis of the interview data. Shenton also stated that dependability in qualitative research is the responsibility of the researcher. I exhibited professional behavior during the sample selection process and provided a thorough analysis of the data to describe the context within which the research was conducted.

Mitigation of limitations. The goal was to obtain a sample size, n, such that $4 \leq n \leq 20$, with the final sample size as close as possible to the upper bound of 20. Initially, 24 participants were identified. Four participants disengaged from the study:

two relocated for a new job, and two were unable to participate due to increased job commitments.

I limited the effect of researcher bias by making the pilot study participants aware of my bias and advising them to focus on phraseology of the research questions to ensure (a) the neutral tone of the semistructured interview questions and (b) the ability of the semistructured interview questions to elicit the desired information from the participants.

Significance

Complex systems, whether software systems or not, are either made up of, or interface to, other systems (Meadows, 2008; Senge, 1990). Stakeholders, in addition to the normal concerns of cost and schedule, need to know whether their software system currently under development is on schedule to interoperate with systems that are either already deployed, under development, or planned for the future. Improved insight into software development progress, cited as needed by Chidamber, Darcy, and Kemerer (1998) and the National Defense Industrial Association (NDIA, 2010), could result in reduced project cost overruns/failures/cancellations, more responsible and effective use of taxpayers' dollars, and improved planning for interoperability.

Despite the growth of, and the many successes in, the software industry, cost and schedule overruns continue. Instead of categorizing failures, a hard look at what program and project managers describe as both positive and negative experiences with metrics could lead to an improvement in the issues identified in the NDIA 2006 report that were not addressed in the NDIA 2010 report.

Significance to Theory

Adherence to measurement theory is lacking in the software industry (Abran, 2010; Meneely, Smith, & Williams, 2012). Although standardization efforts exist at both the international and national levels, consistency does not (Abran, 2010; Texel, 2013). The artifacts (e.g., measures and metrics) used by management to capture and report software development status are lacking, and those currently used to characterize other software development characteristics (e.g., quality and complexity) are faulty. An analysis of managers' experiences reporting and assessing software development status could potentially provide a platform for improving the current state of measurement in the software community.

Significance to Practice

As previously stated, management is often faced with reporting software development completion status with insufficient data. Although reports cite lack of management insight into development status, management does not have the mechanisms necessary to provide that insight. Documenting program and project managers' experiences with current measures and metrics with respect to their need to assess and report status could be a catalyst to begin to examine semantic metrics as an alternative.

Social Change

Managing a software development effort with (a) truisms that may not be implementable, (b) syntactic metrics extracted from program code that are not relevant to completion status, and (c) status reports lacking adequate information with respect to completion status often leave software managers in a difficult position and stakeholders

lacking adequate completion status. Capturing lived experiences with current software management mechanisms represents the first step toward addressing the issues of management ineffectiveness documented in the NDIA 2010 report as well as a first step toward satisfying stakeholders' needs. This research can lead to social change at (a) the individual level for each manager, (b) contractors' organizations, (c) the community of all stakeholders, and (d) the government contracting community as a whole.

The anticipated implications of social change for the embedded software systems discipline represent a combination of Hegelian dialectic, specifically thesis/antithesis/synthesis, and Lenski's focus on evolution (Mueller, 1958; Lenski, 1970). The pattern of overruns and references to lack of management insight into the software development process imply a needed evolution to more robust completion status measurement processes that will be debated by both researchers and practitioners, finally resulting in a synthesis of one or more proposed ways forward.

Vago (2009) viewed social change in terms of five components: (a) identity (what is changing), (b) level (change in hierarchy), (c) duration (length of change), (d) magnitude (defined as a sliding scale from minimum to revolutionary), and (e) rate of change. With respect to my study, analysis of the response data identified a need for an evolutionary change in software development measurement processes. In the software industry, any change of this kind will take at least a decade to implement and will equally affect all levels of the hierarchy. Early adaptors of the challenge will reap the greatest reward: confidence for all stakeholders.

The social implication for all four levels—individual, organizational, stakeholder community, and government contracting community—is that the empirical evidence synthesized from this study supports a need for the identification, development, and employment of new management measurement mechanisms, at an appropriate level of granularity, to capture relevant software development completion status. If the software development process is viewed as a system within a system (Meadows, 2008; Senge, 2010), a change in the software measurement process will affect existing systems within which software development operates, such as quality assurance (QA), and require new software applications to collect and analyze new measurement data. Education and training will be required to ensure that all stakeholders have the requisite knowledge and exposure to a new reporting process. Lastly, individuals, organizations, and the software community, with relevant status data, will have the opportunity to recognize software development status issues earlier. Early detection of factors hindering software development progress will result in cost savings and a reduction of, not dismissal of, overruns. Lastly, improved measurement processes for capturing software development completion data will result in more efficient monitoring mechanisms that will result in improved software measurement technology and more efficient software development production globally.

Summary

Multiple government, commercial, and academic studies have documented the cost and schedule overruns associated with system and software development. Subsets of these studies identify obstacles that impede success. One obstacle repeatedly identified is

lack of management insight into software development processes. The current level of granularity is either too coarse, a line item in a Gantt chart or a process step completed, or too fine, such as source lines of code or algorithmic complexity extracted from program code.

Multiple mechanisms exist to support management in capturing software development characteristics. However, these mechanisms lack a focus on software status. At one end of the continuum of measurements (see Figure 1), there are abstract truisms, Gantt charts, and risk analysis, while at the opposite end, there is a proliferation of detailed metrics. Abstractions do not provide the granularity required to adequately report or assess completion status. Metrics based on detail address characteristics of software other than completion status, specifically quality, complexity, size, and level of effort. There is a gap between the current abstract and detailed management mechanisms, and a second gap between academic literature on, and current practices with, software metrics (Abran, 2010; Day, 2009). Abstractions and detailed metrics do not provide the necessary insight into software development completion status.

I chose a phenomenological research study to explore program and project managers' lived experiences of reporting and assessing software efforts with existing mechanisms. These program and project managers were members of organizations that contract with the government (DOD and non-DOD). With a research effort targeting experiences, both positive and negative, progress can be made toward confirming, or not, and addressing the previously identified gap depicted in Figure 1.

The social implications of my research serve managers; the community of managers, both at the individual and organizational level; the software community in general; and ultimately taxpayers' dollars. The findings confirmed the gap previously identified and dissatisfaction with the de facto mode of operation. Future research might look at semantic metrics, based on knowledge units (KU), along with natural language processing (NLP), to provide a more timely and granular approach to the measurement of software development completion status.

A review of the literature with respect to cost and schedule overruns, syntactic metrics, and phenomenology is included in Chapter 2. The details of the specific operationalization of the phenomenological methodology to be used in this study are specified in Chapter 3. The results of the study are presented in Chapter 4, while conclusions and recommendations for future research are included in Chapter 5.

Chapter 2: Literature Review

To pursue the exploration of managers' experiences of reporting and assessing software completion status, it was necessary to collect and synthesize current research that (a) documented overruns, (b) referenced oversight as a contributing factor or obstacle to success and (c) identified, and supported the gap in current measurement mechanisms to address software completion status. Following two initial sections that contain a brief description of the literature search strategy and a concise synopsis of current research, the third section contains the literature review. The literature review is comprised of five subsections as follows: history of overruns, software management truisms, software metrics, automated tool support, and metric validation. Lastly, an overview of the selected research approach, phenomenology, is provided, followed by a chapter summary.

Literature Search Strategy

The literature review sources originated from multiple research databases, primarily IEEE Xplore, ACM Digital Library, and Business Source Complete. I searched peer-reviewed journals using the following Boolean search expressions:

- *software* and *metrics*
- *object* and *oriented* and *metrics*
- *software* and *management* and *metrics*
- *software* and *metric* and *validation*
- *software* and *project* and *overruns.*

Citation-chaining on relevant articles obtained from the search led to the identification of additional articles. Citation-chaining continued until authors were repeatedly cited. Those authors' names then became the input to additional searches by author. Additionally, I conducted searches on known seminal authors in the software community. Those seminal authors and their specific software-related subdiscipline, or area of expertise, are identified in Table 5.

Table 5

Seminal Authors

Seminal authors	
Discipline	Authors
Software management	Booch (1996, 2005, 2007) Brooks (1995) DeMarco (1982, 1995, 1997) DeMarco (2003) Humphrey (1987, 1988, 1989) Jones (1995, 2004, 2008, 2010) Yourdon (2004)
Software metrics	Abran (2010), Fenton (1997)
Object-oriented software metrics	Abreu (1994, 1995, 1996) Chidamber (1994) Li (2000) Lorenz (1994)
Validation	Briand (1996) Kitchenham (1995) Weyuker (1988)

Note. References, formatted to APA sixth edition, are contained in the reference list. Only the first author is indicated in this table.

Conceptual Framework

Figure 3 represents a concept map highlighting the coverage of the major concepts, and relationships among them, in this literature review. These concepts have

either a direct or indirect relationship with each other, and all affect the ability to gain insight into software development completion status.

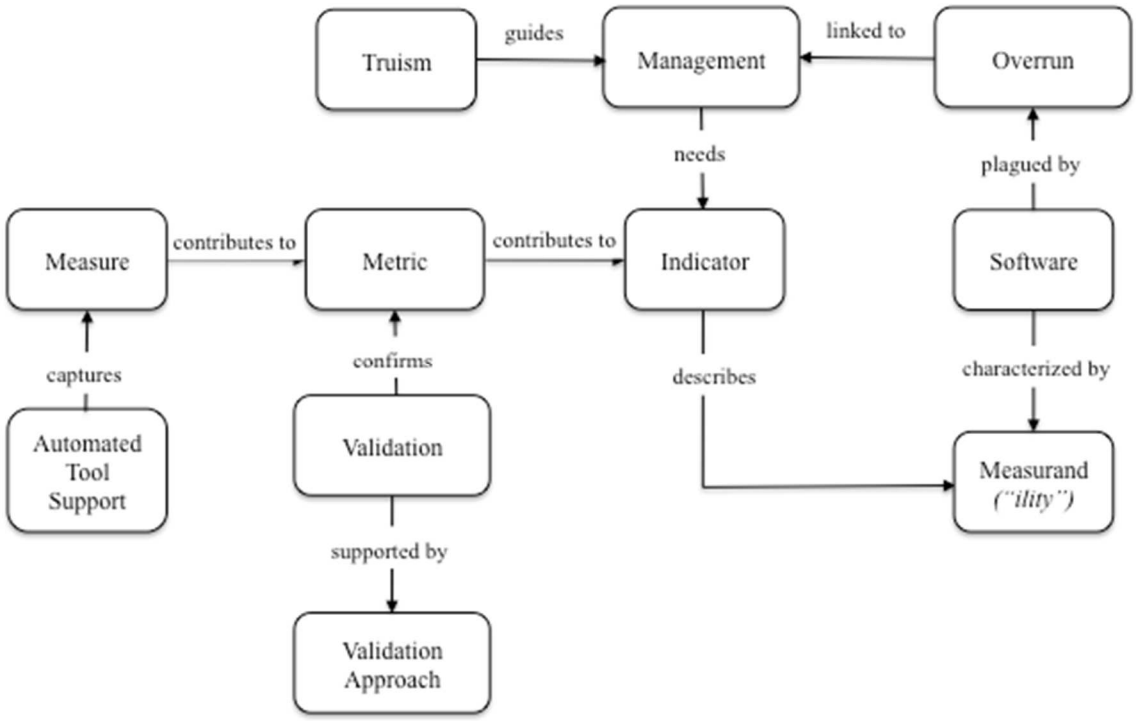


Figure 3. Literature review concept map (figure created by P. Texel using Microsoft PowerPoint).

The three concepts located at the top right quadrant of Figure 3 indicate that overruns are irrevocably linked with systems and software efforts and plague software-intensive system development efforts. Multiple government and commercial reports, as well as academic studies, have documented overruns

and identified many common issues affecting successful deployment. Lack of management insight into software development status is one repeatedly documented issue, even at the presidential and Congressional levels.

Management does have mechanisms to support the monitoring of development efforts, but these mechanisms, ranging in level of detail from abstract truisms to detailed syntactic measures/metrics, represent two ends of a continuum. Abstract truisms are difficult to implement and detailed measures/metrics are (a) ambiguous and open to interpretation, (b) subjected, or not, to any of diverse validation frameworks, and (c) extracted from program code by various automated software tools that are based on different counting rules. These detailed measures/metrics become input parameters to cost and level of effort software estimation tools whose results are used to (a) negotiate business contracts and (b) measure many characteristics of software, but not software development progress. Management works with estimates based on estimates, ambiguous measures/metrics, diverse validation approaches, and a lack of useful progress indicators. The literature review in this chapter validates this perspective from a technological viewpoint, and the research study focused on exploring managements' lived experiences with this phenomenon.

Along the horizontal path in the center of Figure 3, the terms *measure* and *metric* are ambiguous, misused, and used interchangeably in the industry (Texel, 2013). As shown in Figure 3, an indicator of a software *ility* (e.g., quality, dependability) is based on metrics, and metrics are based on measures. Management needs indicators to manage; however, indicators fail when based on faulty measures or metrics (Texel, 2013). A subsequent subsection contains discussions of the faulty nature of measures and metrics.

Additionally, measures and metrics need to be validated, as is done in other industries (Abran, 2010). However, validation of software measures is shown to be as elusive and ambiguous as measures and metrics. Three main validation mechanisms ground the software metric validation subdiscipline today: set theory (Briand, Morasca, & Basili, 1996), properties (Weyuker, 1988), and a measurement model to support the identification of elements associated with measurement and their validation (Kitchenham, Pfleeger, & Fenton, 1995). Multiple additional validation models exist, and the results of a recent meta-analysis indicated that the same lack of consistency with respect to measures and metrics is found within the metric validation subdiscipline (Meneely et al., 2012). Despite that lack of consistency, Gandhi and Bhatia (2012) added two new reuse metrics applicable to OO software and validated those metrics against Weyuker's properties.

As indicated by the concepts in the left quadrant of Figure 3, software tools produce measures and metrics by accessing and data mining software program code. The literature supports that automatic counting tools use different algorithms to extract software measures and metrics and, when executed on identical program code, the degree of variation between results is staggering. Additionally, when different projects within a division of an organization use different counting tools, corporate comparison of multiple projects is not meaningful, and management again lacks support data.

Literature Review

My review of the literature revealed that (a) ambiguous measures lead to failed indicators, (b) diverse validation approaches exist, and (c) the existence of multiple ways of extracting measures from program code leads to diverse results. The impact is a lack of rigor within the software engineering community with respect to *metrology*, the study of measures. Consequently, managers are left to steer programs and projects without a valid compass. Table 6 represents a mapping of the topics previously identified in the concept map, Figure 3, to the five subsections of this literature review section.

Table 6

Literature Review Subsections Mapped to Concept Map Topics

Literature review subsections mapped to concept map topics	
Section title	Figure 3 concepts
Literature Review	Management, Overruns, and Software
History of Overruns	Management, Overruns, and Software
Software Management Truisms	Management
Software Metrics	Measure, Metric, Indicator, and Measurand
Software Metric Validation	Metric Validation, Validation Approach
Automated Tool Support	Automated Tool Support

History of Overruns

Despite many successes within the system and software industry, cost and schedule overruns continue to plague development efforts (Accenture, 2014; Howell & Dinan, 2014). When an airplane crashes or a train accident occurs, the National Transportation Safety Board (NTSB) conducts a lengthy and thorough investigation and

documents findings in an effort to avert future disasters. When a bridge collapses, the appropriate local, county, or state authority investigates the collapse to determine the cause, or causes, that led to the collapse with the sole purpose of avoiding similar collapses in the future. Numerous reports, spanning decades, document summaries of system and software cost overruns, schedule overruns, and project failures. Initially introduced by De Marco (1995) and confirmed two decades later by Eveleens and Verhoef (2010), post mortems are very rarely conducted on failed software projects, and overruns continue today.

Multiple sources of cost and schedule overrun reports exist: government agencies, commercial organizations, and academia. The commonality in the reports is the focus on overruns. The diversity represented in the reports is the software application focus (e.g., defense, commercial) and software program types (e.g., ERPs, weapon systems). This commonality and diversity have resulted in broad, yet uncoordinated, reports that are chronologically synthesized in the following paragraphs.

As previously stated in Chapter 1, the Standish Group periodically reports overruns and percentages of projects that were successful, challenged, or failed. The percentages in Table 1 reflect a slow increase in the number of successful projects. However, Eveleens and Verhoef (2010) and Hofbauer et al. (2011) challenged these findings based on faulty definitions and transparency of process.

An analysis of 250 projects over a 10-year period from 1995 to 2004 representing information systems (IS), systems software, outsourced projects, and defense applications

indicated that 25 (10%) of the projects were successful; 50 projects (20%) experienced overruns less than 35% of the initial contract value, and 175 (70%) experienced either delays or overruns, or failed completely (Jones, 2004). A 2008 report by the European Services Strategy Unit (ESSU) documented the results of a meta-analysis of 105 public sector information and communication technology (ICT) failures spanning a decade, and listed the following key findings: (a) actual budget for 105 projects was £29.5 billion and cost overruns equaled £9 billion, (b) 57% of contracts experienced overruns, and (c) the average percentage cost overrun was 30.5% (Whitfield, 2007). A summary of a 2009 GAO report on weapons systems initiatives documented almost $296 billion in overruns and 66.7%, specifically 64 out of 96 projects, were overrun (Galorath, 2011). A 2010 GAO report (GAO-10-1059T) documented a $6.9 billion overrun for 10 ERP systems (USDOD, 2010). In April 2011, the Defense-Industrial Initiatives Group of the Center for Strategic and International Studies (CSIS), citing a 2010 GAO study, stated that 98 Major Defense Acquisition Programs (MDAPs) were, in total, $402 billion overrun and experienced an average delay of 22 months (Hofbauer et al., 2011). Finally, A 2012 Department of Defense (DOD) Inspector General (IG) report (DODIG-20120111) identified a postponement of a major Air Force accounting system upgrade from October 2009 to April 2017 with a corresponding increase in cost of $1.78 billion (USDOD Inspector General [USDOD IG], 2012). Whether nationally or internationally, whether for commercial or defense applications, annual software cost overruns are irrevocably linked to the software industry.

The problems with cost overruns are well known at the Congressional and presidential levels. Emerson, Chair of the Subcommittee of Financial Services and General Government Appropriations, a subcommittee of the Committee on Appropriations, stated,

> We do not have a great track record in this government on IT, and I can't begin to tell you how many, probably billions if you add it all up, of dollars have been spent. And it has not been well spent whatsoever. ("Financial Services," 2011, p. 58)

In a Senate nomination hearing for the appointment of General Dempsey to the position of Chief of Staff of the U.S. Army, Senator McCain (R-AZ) cited a 2011 Decker-Wagner Army review of 2010 Army acquisitions that concluded, "between $3.3 and $3.8 billion of the Army's research and development budget has been wasted per year, since 2004, on programs that were subsequently cancelled" ("Nominations before," 2011a, p. 106). In that same hearing, Senator Lieberman (D-CT) asked General Dempsey for "initial thoughts on how the Army can best rise to what I describe as the software challenge, particularly the element of leadership" ("Nominations Before," 2011b, p. 109).

President Obama, in a March 2009 press release calling for more accountability with respect to monitoring government contracts, stated,

> It is essential that the Federal Government have the capacity to carry out robust and thorough management and oversight of its contracts in order to achieve

programmatic goals, avoid significant overcharges, and curb wasteful spending. (The White House, Office of the Press Secretary, 2009, para. 5)

Obstacles to success. Obstacles that impede the development and deployment of software-intensive systems within cost and schedule constraints have been well documented (DeMarco, 1995b, 1995c, 1995d, 1997; USDOD IG, 2012; Yourdon, 2004). These obstacles include requirements volatility (Dev & Awathi, 2012; Hull et al., 2011), inadequate estimation techniques (Singh et al., 2011), and ineffective management (NDIA, 2010; The White House, Office of the Press Secretary, 2009; Whitfield, 2007). Requirements volatility (Dev & Awathi, 2012; Hull et al., 2011) and poor estimation techniques (Singh et al., 2011), although cited as serious issues impacting successful development of software projects, were not the focus of this research and are not addressed further; the issues that affect management effectiveness were the focus of this research.

With respect to the issues that management faces, multiple mechanisms exist to support management in monitoring the development and deployment of software systems. One set of mechanisms includes items such as Gantt charts, risk analysis, variations in cost and schedule, and resource allocation. Another set of mechanisms includes software metrics extracted from program code, for example SLOC and CCM. As previously introduced, and illustrated in Figure 1, the two sets of mechanisms lie at opposite ends of a continuum of mechanisms. Subsequent subsections of this chapter contain more detailed exploration of these two approaches.

However, there are too many metrics, initially introduced by Chidamber et al. (1998) and later supported by Abran (2010). According to Abran, the proliferation of publications on metrics and the sheer volume of metrics suggest the necessity to focus on whether the metrics are achieving the intended goal, to support management in monitoring software development progress. An additional concern is the gap that exists between practitioners and researchers, as evidenced by the lack of consistency and acceptance within the software engineering discipline.

Addressing obstacles to success. To summarize, cost and schedule overruns continue today and a consolidated agency to analyze issues, propose solutions, and enforce adherence to guidelines for success is lacking. There is one exception, the Carnegie Mellon University (CMU) Software Engineering Institute (SEI), herein referred to as CMU-SEI. Established in 1984, the goal of the CMU-SEI focused on the establishment of objective guidelines to assess software development contractors' maturity to perform on a contract. The primary goal of the CMU-SEI was to foster the growth of the software engineering discipline leading to improved development of, and resulting quality of, software systems. Unfortunately, the CMU-SEI reports, although funded by the DOD, do not represent official DOD positions and corporate participation is voluntary.

Following initial work characterizing the maturity of a software development process (Humphrey, 1987, 1988, 1989), the CMU-SEI issued multiple reports that (a) documented the capability maturity model (CMM) model, or framework, for evaluating

the maturity of a software development process on a scale of 1 (ad hoc) to 5 (mature: CMU/SEI-93-TR-024: Paulk, Curtis, Chrissis, Weber, 1993), (b) identified key practices for steps within the model (CMU/SEI-93-TR-025: Paulk, Weber, Garcia, Chrissis, & Bush, 1993), (c) documented options in CMU/SEI-192-TR-020 for counting lines of program code (Park, 1992), and (d) listed options for measuring object-oriented software (CMU/SEI-95-TR-002: Archer & Stinson, 1995).

The SEI-CMM framework provided an objective, grounding structure for managers to lead an organization in identifying the current level of software process maturity and assisting in the evaluation of software development contractors who respond to Request For Proposals (RFPs; Carnegie Mellon University Software Engineering Institute [CMUSEI], 1995). The hypothesis was that a concentrated focus on implementing and improving a software development process would increase (a) the probability of successful software deployments and software quality and (b) provide managers with guidelines that support improved insight into the process (Paulk et al., 1993).

The CMU SEI CMM continues today as the CMM Integrated (CMMI), a baseline for developing/evaluating software development processes, despite disagreement as to whether that hypothesis has been supported. First, the model has a limited scope, specifically targeting a single project (Paulk et al., 1993). Second, the model does not address multiple or distributed projects (Paulk et al., 1993). Third, the model does not address alignment with business goals and objectives (Basili et al., 2007).

Now having evolved into the CMMI and managed by the CMMI Institute, the CMMI by design is conceptual in nature, providing guidelines for what process improvement activities need to be addressed but leave the implementation of the abstractions to the implementer. An example of this *what versus how* conundrum is illustrated in Figure 4.

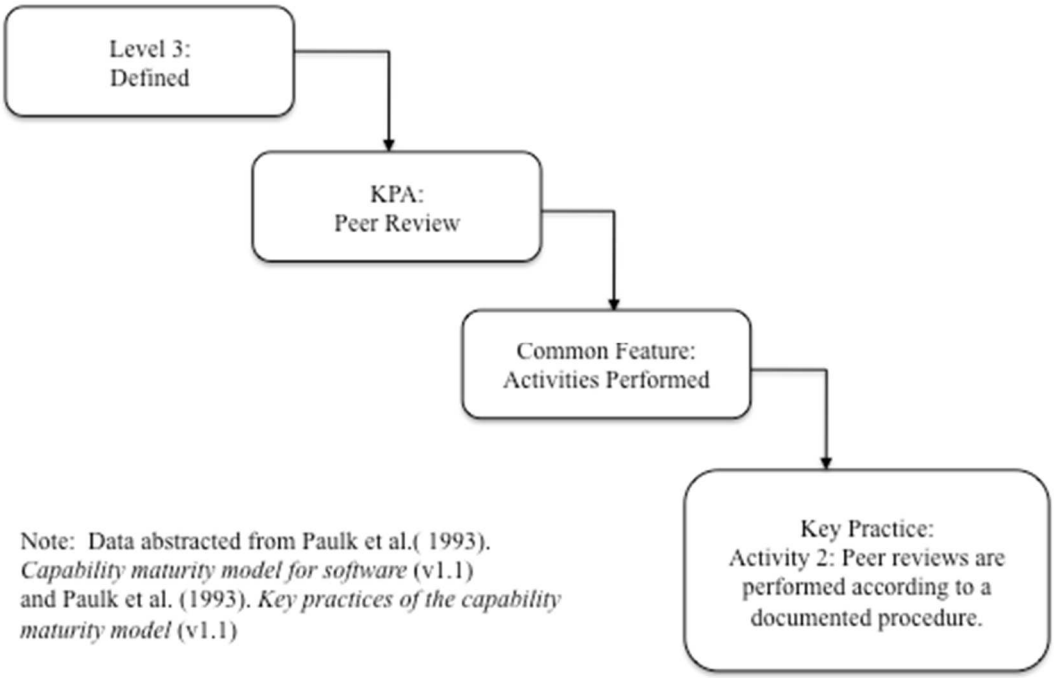


Figure 4. Sample key practice, Level 3, KPA peer review (figure created by P. Texel using Microsoft PowerPoint).

Although Paulk et al. (1993) defined a peer review as a process to be conducted and documented, neither implementation of the peer review process nor the requisite components of supporting documentation are specified but rather left to project specific implementation. Leaving the implementation up to a project allows a project to customize a specific implementation according to resource and time constraints.

However, as a result of this *what versus how* conundrum, an organization with multiple projects could have multiple instantiations of the CMMI and miss the opportunity to evaluate software projects at a corporate level.

Lastly, the latest CMU-SEI Maturity profile dated September 2012, summarizing CMMI certification data, indicated that of 5,069 organizations reporting, 80.3% of the organizations reporting certification were from organizations outside the United States and 19.7% were from the United States (CMUSEI, 2012). Unfortunately, acceptance of the CMMI model by non-U.S. countries far exceeds acceptance within the Unites States and the United States lags China and India in the number of Level 5 certifications (CMUSEI, 2012).

In summary, the CMU-SEI CMMI provides a substantive and valuable framework supporting a concentrated effort to improve or evaluate software development processes and products. However, embracing the framework is voluntary and time-consuming. Nor does the CMMI framework directly address the detection of software development progress but rather outlines an overall framework defined by key process areas (KPAs) and the key activities to perform within each KPA. Measurement and the use of metrics to support measurement are recommended. However, which specific metrics to use are organizationally, and even project, dependent.

As previously introduced, and graphically depicted in Figure 1, management mechanisms to monitor software development status exist on a continuum, from abstract truisms to detailed measures and metrics. There is a gap in the literature with respect to

the middle of the continuum when considering software development completion status. Subsequent subsections contain descriptions of the two ends of the continuum: their pros and cons, and their relevance to the capture of software development completion status.

Software Management Truisms

DeMarco (1995d), as a result of researching the high cost of software, offered best practices that if followed, would support the reduction of costs associated with software development. A representative subset of these best practices included (a) conduct post-mortems for both successful and failed software efforts, (b) keep staff involved and allow their voices to be heard, (c) improve quantitative management practices, and (d) make each day count. In a subsequent fictional treatment of a software manager in charge of an ideal software project, the manager was followed throughout the project as multiple obstacles were encountered that, upon hindsight, led to the identification of additional truisms (DeMarco, 1997). A subset of those truisms, still true today, is contained in Table 7.

Not quite two decades later, Jones (2010), a seminal author with respect to software program management, identified 50 best practices. Compiled over 30 years, these best practices encompass both managerial and technical topics. Examples include (a) mitigating effects of layoff due to downsizing, (b) reviewing architecture and program code, (c) establishing configuration management, (d) establishing quality assurance, and (e) tracking project milestones (e.g., requirements review, project plan review, cost

Table 7

Sample Truisms

Sample management truisms		
#	Focus	Truism
1	Personnel	Hire smartly, assign carefully, motivate well, listen
2	Team	Motivate, support, ensure cohesiveness
3	Defensive strategy	Contain failures, cut losses. Time lost at beginning of project just as devastating as time lost towards the end of a project
4	Models	Use & improve models
5	Politics	Play well
6	Change	Expect change, be flexible

Note. Data abstracted from DeMarco (1997). *The deadline: A novel about project management.*

estimate review, deployment plan reviews, code reviews, and system and acceptance test plan reviews). These are all valid, necessary, and contribute to the overall management of software projects. However, once again, these 50 best practices do not address the incremental completion status of the software development effort.

Summary: Software management truisms. Unfortunately, although management truisms and best practices represent valid, useful, contextual information based on experience, they do not contribute to the capture of software development completion status, nor was that the stated intent. As documented by Jones (2004), software projects can fail for multiple reasons, and succeed for only a few. One common thread is management. The next subsection contains a description of the issues with detailed syntactic software measures and metrics and their inadequacy to provide management with necessary completion status.

Software Metrics

The focus of this subsection is on the inability of the existing software metrics framework to provide insight into software completion status. As previously introduced and illustrated in Figure 1, software measures and metrics lie on the opposite end of the management mechanism continuum from truisms. First, however, is a brief history of software metrics to show how the industry arrived at its current state.

The history of software metrics is intertwined with the evolution of programming languages and software development processes. As shown in Table 8, the first software metric, dating back to the era of assembly language programs (1950s), was a count of the number of source lines of code (SLOC) that was used as both a software development estimate and progress metric. Monolithic in nature, assembly language programs consisted of assembly language statements that execute sequentially. SLOC was easy to get, easy to use, and all that existed.

Since the 1950s, metrics have increased in number, as supported by Appendix A. However, the increased focus on object-oriented systems has not migrated away from counts of program code elements and calculations based on those counts. The software engineering community continues to count. Additionally, more metrics are emerging for agile software development (Aktunc, 2012; Farid & Mitropoulos, 2013; Tabib, 2013; Tarhan & Yilmez, 2014), CBSS (Abdellatief et al., 2013), and semantic metrics (Chandrika, Babu, & Srikanth, 2011; Gall et al., 2008; Ma et al., 2011; Stein et al., 2009).

Table 8

Evolution of Software Metrics

Evolution of software syntactic metrics		
Time period	Programming language/process	Example metrics
1950s	Assembly, FORTRAN	SLOC
1960s	BASIC	SLOC, NOM
Late 1960s & 1970s	C, PASCAL SMALLTALK	McCabe CCM, Halstead
1980s	C++, Ada83	Counts of language constructs
1984 1987	CMU SEI established 1st CMU SEI CMM	
1990s	UML, Java, Ada 95	CMU-SEI Metrics, OO Metrics
21st Century	C# VisualBasic.net Ada 2012 Agile CBSS	100s of metrics and climbing Agile process metrics CBSS metrics

However, there are many issues with syntactic measures and the inability to support monitoring software completion status. First, because syntactic measures are not available until program code exists, they do not permit insight into the phase(s) of a SDLC preceding design and code. Second, syntactic measures are based on simple counts and arithmetic calculations on those counts. A relationship between the semantics of the problem space and the semantics of the solution space is not supported (Gall et al., 2008; Stein, 2004). Third, definitions of syntactic measures are open to multiple interpretations resulting in the lack of consistency in the community, as well as

difficulties with meta-analysis and meaningful discussions (Texel, 2013). Fourth, as previously stated, there are too many syntactic metrics and concurrently a lack of acceptance by software practitioners (Abran, 2010). This state of measures and metrics is reminiscent of a question posed by DeMarco (1995b): "Are we doing best what we shouldn't be doing at all?" (p. 42). That same question can be asked today, almost 20 years later. The software industry has made extraordinary progress from the 1950s to the present day: programming languages have evolved, systems are larger, more complex, and provide ever-increasing functionality. Measures and metrics have increased in number and scope, but remain unable to support the detection of current completion status. Once again, management lacks valid measures or metrics to monitor software development completion status.

Two of the most commonly used syntactic measures are SLOC and complexity (Briand et al. 1996; McConnell, 2010; Jones, 2010). The intended purpose of the following discussions on SLOC and complexity is to provide a clear demonstration as to the lack of clarity of the meaning of these two most commonly used measures. An additional purpose is to illustrate the kinds of flaws associated with many syntactic metrics in general.

Source lines of code (SLOC). SLOC became the de facto metric for software progress and decades later SLOC is still used, despite a current lack of relevance to software development progress (Booch et al., 2007). Lorenz and Kidd (1994) recommended that SLOC not be used at all. SLOC has also been called statistically

unreliable (Schofield, 2005), useless and harmful (Abran, 2010), and hazardous (Jones, 2010).

SLOC represents a count of the number of lines of code in the software entity being measured, whether module, subsystem, or system. Researchers have addressed several issues with SLOC, specifically the ambiguity of SLOC itself, the impact of rework on SLOC numbers, inconsistency in counting tools, and the inability of SLOC to reflect knowledge. Subsequent subsections contain discussions of these issues, as well as the identification and discussion of partial benefits associated with SLOC.

Ambiguity and code size variation. The meaning of SLOC is ambiguous, meaning that different interpretations of SLOC exist (Jones, 2008; Park, 1992; Schofield, 2005). Multiple reasons exist for variations in code count. Two frequently identified sources of variations in code count are programming style and programming language (Etzkorn, Gholston, & Hughes, 2002). Studies of variations in code count date back to DeMarco (1995c). Analysis of data from 16 programmers, producing code to the same statement of requirements, exhibited differences in programming style, resulting in code count variations between 100 and 500 lines of code or more.

A decade later, a longitudinal study analyzed program size data collected from six iterations of a Personal Software Process (PSP) class where each class consisted of 10 students who wrote nine programs each for a total of 540 programs (Schofield, 2005). Variation between minimum and maximum program size, by programming language, as

well as the range of values for the mean number of lines of code and associated standard deviation, are summarized in Table 9.

Table 9

Effect of Programming Language on Code Count

Code count by programming language				
Programming language	SLOC % variation		SLOC Mean/*SD*	
	Min	Max	Min	Max
PL1	150%	467%	97/47	184/71
PL2	381%	2,223%	89/73	122/97
PL3	415%	1,794%	59/50	146/82

Note. Data extracted from Schofield (2005). *The statistically unreliable nature of code.*

One conclusion Schofield (2005) drew from the data targeted the wide variation of data points and indicated, "the counts are practically useless in the best case, harmful and misleading in the worst cases" (p. 29). Having indicated that the SLOC metric did not keep pace with changes in software development processes, Jones (2008) stated that the usefulness of SLOC had deteriorated to such point that SLOC had become "actually harmful" (p. 10).

SLOC counting rules. Because of the ambiguity of the term SLOC, multiple ways to count SLOC exist. For example there are 30 potential definitions for code count when considering just two options: line type and comments. There are two kinds of line type— physical and logical lines—and three possible kinds of comments—in-line, block (including single line), and header.

Focusing on just the number of ways to count comments, there are 8 possible ways to count comments with the options identified above, including not counting comments at all. The number 8 is calculated using the mathematical formula for calculating the number of possible combinations of items where the order of the items is not significant. The formula accounts for all combinations of the comment types taken none at a time, one at a time, two at a time, and three at a time as follows:

$$C_0^{\,3} + C_1^{\,3} + C_2^{\,3} + C_3^{\,3} =$$
$$1 + 3 + 3 + 1 =$$
$$8$$

Considering that these 8 ways are applicable to counting either physical or logical lines of code, there are now a total of 16 potential ways to count code. Now add the option of including, or not, blank lines. With this additional option the updated total number of possible counts increases to 16 x 2 or 32.

Now consider language specific issues, for example JavaDoc comments, customized comments to provide web-enabled code documentation for Java Programs. The option to include, or not, JavaDoc comments in a code count increases the number of possible counts to 64. Other languages do not have the JavaDoc comment capability, thus illustrating variations in count among programming languages. This latter fact indicates the complexity involved in providing code counts for software systems developed in multiple languages.

These differing counts point to the ambiguity of SLOC due to multiple interpretations and implementations with respect to code counting rules. Consistencies in

code counts within one project permit SLOC to be compared over time, however consistency in SLOC across projects cannot be accurately compared unless the programming style, programming language, and counting rules are identical.

To facilitate consistency and decision making with respect to SLOC, a report issued by the Software Engineering Institute (SEI: CMU/SEI-92-TR-020) included a Checklist for Source Statement Counts (CSSC) that identified 66 potential code count variations with an additional eight more language dependent counts (Park, 1992). The CSSC did not include potential variations of options as described above. The real number of potential ways to count SLOC remains elusive. Taking into consideration that CMU/SEI-92-TR-020 consists of over 170 pages plus five Appendices focused on measuring software size, the real number of ways to count SLOC remains elusive.

Rework. In addition to the above issues, SLOC counts may, or may not, include lines of code due to rework (Mozoroff, 2010). In an analysis of three projects, Mozoroff found that the amount of code written, but not included in a software deliverable, ranged from 19% to 40%, and that reworked code count was larger than the count of code that was added. Consequently SLOC did not capture true programmer level of effort, which increased and decreased reflecting code added, modified, or removed; a natural progression of rework.

Relevancy of SLOC to knowledge. Counting lines of code does not reflect knowledge gained (Armour, 2004). An increasing SLOC count over time indicates that software size is increasing, but increasing SLOC does not measure problem domain or

programming knowledge. A specific user operational error may not be supported in the code or perhaps nested if statements are used in place of a short circuit control form.

Equally important to measuring knowledge is measuring the lack of knowledge. Gaining knowledge is what takes time and there is no measure to measure what is not known (Armour, 2004). Armour indicated the industry needed to "come up with a unit of knowledge or a way of counting it" (p. 24).

SLOC as basis for estimates. The issues with SLOC, a seemingly simple count of the number of lines of code in a software program, serves as an example that a simple syntactic measure is anything but a simple syntactic measure and brings into question the validity of cost and resource estimates based on SLOC, which is the basis for many software cost and estimation tools. Two of the most recognized estimation models in the industry are function point analysis (Jones, 1995) and the constructive cost model (COCOMO) according to Boehm et al. (2000). Both estimation models require SLOC as an input to the estimation algorithm.

SLOC is a highly controversial subject in the software industry. Consistency is missing in the (a) definition of SLOC, (b) counting rules, (c) counting tools, (d) tools that utilize SLOC for estimation and forecasting, and (e) tools that utilize SLOC for estimating level of effort. Yet despite these issues, SLOC is still used to estimate software project schedule and resource requirements. Despite the statement that "total size alone is inadequate as a progress measure," program code size is still used to report development progress (Park, 1992, p. 76). Additionally, Park stated that "the inadequacy

of total counts as management metrics is compounded further" when copied code is not considered (Park, 1992, p. 76).

Complexity. SLOC is only one of the many measures and metrics that are the subject of controversy. A second very common measure associated with program code is complexity. There is no consistency within the software engineering community on the definition of complexity. Software *c*omplexity is alternatively defined as efficiency (Fenton & Pfleeger, 1997), relationship between operators and operands (Halstead, 1977), number of linear paths through program code (McCabe, 1976), statement count (Weyuker, 1988), weighted methods per class (WMC) for object-oriented software (Chidamber & Kemerer, 1994), and knowledge (Etzkorn & Delugach, 2000). Industry consistency on the meaning of complexity remains as elusive as industry consistency on how to count source lines of code.

Jones (2008) introduced a software complexity taxonomy representing a comprehensive and valid perspective on complexity. This taxonomy is another example of the ambiguity that arises when discussing complexity and the difficulty for managers to choose which is valid for an organization or project. Twenty kinds of complexity, listed in Table 10, were identified. This taxonomy is again an indication that software engineering is simply not aligned with other engineering disciplines with respect to the science of measurement (Abran, 2010; Jones, 2010).

Table 10

Software Complexity Taxonomy

Software complexity taxonomy			
Taxon	Definition	Taxon	Definition
1. Algorithmic	The difficulty of the computational solution	11. Harmonic	Waveforms and Fourier transforms
2. Computational	Effort in length and time to compute an algorithm	12. Syntactic	Grammatical structure
3. Informational	Representation of data structure	13. Semantic	Knowledge based
4. Data	Number of data items and relationships	14. Mnemonic	Memorization
5. Structural	Pattern representation	15. Perceptional	Visual appearance
6. Logical	Boolean expression	16. Flow	Data flow among modules
7. Combinatorial	Permutations and combinations	17. Entropic	Decay and disorder
8. Cyclomatic	Nodes and edges of graphs	18. Functional	User operational capability
9. Essential	Nodes and edges of reduced graphs (redundancy removed)	19. Organizational	Grouping
10. Topologic	Rotations and folding (mathematical)	20. Diagnostic	Errors

Note. Extracted from Jones (2008). *Applied software measurement: Global analysis of productivity and quality.*

An additional approach to complexity proposed that different definitions of complexity were needed at different phases of the SDLC (Hendersen-Sellers, 1996). Complex software requirements have characteristics distinct from complex algorithms that, in turn, have characteristics distinct from complex testing required to validate implementation of software system requirements. Hendersen-Sellers pointed out the necessity to migrate away from structural, or procedural, complexity towards semantic complexity, or complexity of knowledge.

When looking at just one measure, of the many complexity measures included in Appendix A, there is disagreement. Chidamber and Kemerer (1994) introduced an

object-oriented metric suite. Today that suite remains a stable and well-respected set of software engineering metrics. However, a subset of the proposed metrics is the subject of differing opinions.

One such complexity measure, specifically the *number of methods* (NOM), has been critiqued with respect to the lack of addressing how some methods are counted, for example overloaded methods, overloaded operators, and inherited methods (Churcher & Shepperd, 1995). This ambiguity in defining NOM has a direct effect on another metric that uses NOM in its algorithm, specifically *weighted methods per class* (WMC; Churcher & Sheppard, 1995). Although agreeing in principle to the need for clear definitions, Chidamber and Kemerer (1995) countered with a guideline: methods designed and contained in a class should be counted and indirect methods (e.g., through inheritance) should not be counted. Both sets of authors' calculations on the same C++ example resulted in the same value for complexity when using their own definitions: an anomaly. The take-away from this example is once again, interpretations and opinions abound. There is no consistency within the community consequently reliance on even the simplest measures (e.g., SLOC and NOM) is risky. Despite this continuing controversy over interpretations, definitions of complexity continue to morph, as supported by emerging research in semantic metrics (Chandrika et al., 2011; Gall et al., 2008; Ma et al., 2011), agile metrics (Farid & Mitropoulos, 2013), and CBSS metrics (Abdellatief et al., 2013).

Automated Code Counting

Continuing the ambiguity associated with SLOC and complexity is the number of commercially available automated counting tools (i.e., software packages) that generate SLOC. In 2004, over 75 software packages providing counting capability were identified (Jones, 2004). One limitation to these tools is the fact that automatic code counting tools can only count what is represented in the syntax, or grammar, of the program code. Counting is solely based on parsing the program code. Another limitation of these tools is the differing interpretation of counting due to (a) the interpretation of measures and (b) the way the measures are to be counted.

One study analyzed three different open source projects using three different code-counting tools with results reported in Figure 5 (Nguyen, Deeds-Rubin, Tab, & Boehm, 2007). The data indicates that consistency in counting code is lacking and the variation between counts is wide.

Estimation. SLOC is often used as an input to software cost, schedule, and resource estimation software packages (Armour, 2004; Nguyen et al., 2007). Questionable is whether the cost and level of effort estimates produced are valid when SLOC, with so many interpretations, is provided as input. Furthermore, the number of lines of code is not known when estimating software cost, schedule, and resources because the coding phase has not begun. Lines of code do not exist to be counted when estimation takes place (Armour, 2004). Estimates are based on estimates with lack of agreement within the industry with respect to the meaning of the estimates used as inputs.

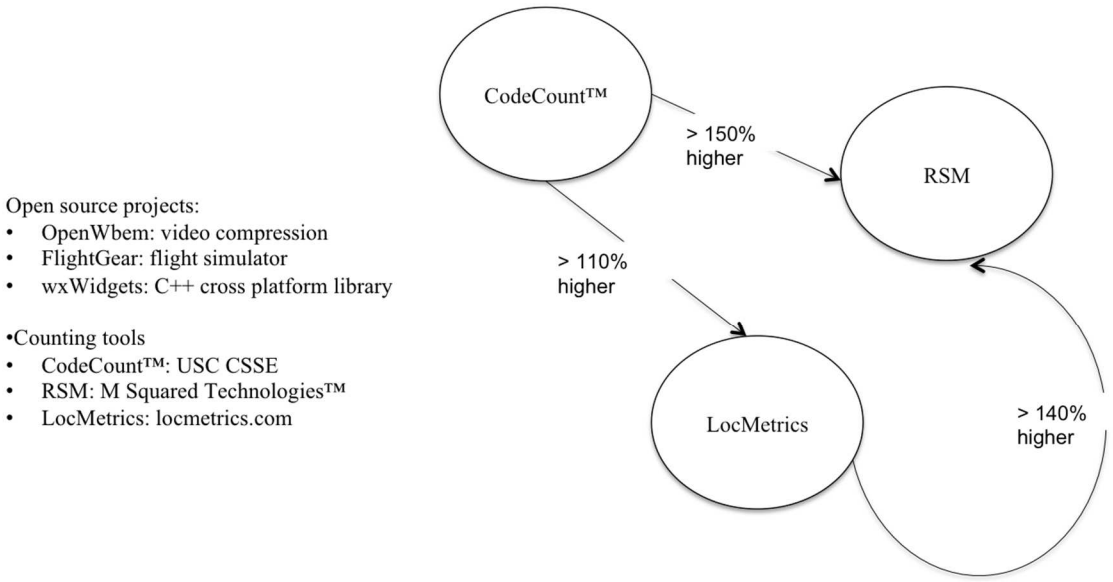


Figure 5. Code count variations: Three projects, three tools (figure created by P. Texel using Microsoft PowerPoint 2011).

In the mid 1970s, function points (FPs) were offered as an alternative to SLOC for estimation purposes and continue today as a mainstay for estimating software size. A FP is defined as a subset of functionality from the user's point of view (Jones, 1995). The number and type of function points are used as input to multiple cost estimation models. FP analysis has been, and will continue to be, a mainstay in estimation approaches. However, the estimation algorithm includes a step that maps FPs to SLOC (Jones, 1995). Stated differently, SLOC count contributes to the estimates based on FPs; an estimate is being used to generate another estimate.

Additional cost estimation techniques, whether estimating size of procedural modules or object-oriented classes, rely on estimated code counts (Armour, 2004). With

the definition of SLOC and the counting tools that produce SLOC varying so widely, solid reliance on estimates produced by cost estimation models is risky, as illustrated in Table 11. Table 11 represents the continuation of the analysis of the data underpinning the counting variations identified in Figure 3. The wide variation in estimated level of effort is concerning when considering a contractual obligation based on these estimates.

Table 11

Variation in % Level of Effort: Three Tools, Three Projects

	Code count percentage variation		
	CodeCount™	RSM	LocMetrics
LOE* (person months)	127	85 (67%)**	112 (88%)

*LOE = Level of Effort
**The percent represents percent variation from the CodeCount™ estimate.
Note. Data extracted from Nguyen et al. (2007). *A SLOC counting standard.*

Summary: Software metrics. There is one constant in the software measurement community, specifically the lack of consistency on (a) definitions, (b) quantification of measures (Abran, 2010: Jones, 2010; Meneely et al., 2012), (c) counting algorithms, (d) estimation tools that utilize measures to predict software size, (e) cost, (f) schedule, and (g) level of effort. Two of the most common software measures, SLOC and complexity, do not have consistent definitions within the software engineering discipline. Additional software measures replicate this lack of consistency with respect to definitions and leave management without standards or guidelines for comparisons. The problem is compounded when multiple programming languages are used to implement a development effort. This state of measures and metrics is embarrassing to

the software community when multiple programming languages are used and should be viewed as “professional malpractice” (Jones, 2010, p. 112).

Software Metric Validation

The current state of software metrics validation, lack of consistency, mimics the current state of software measures and metrics previously identified, and the tools that count program code previously discussed. There are a variety of approaches to validating software metrics and they range from a rigorous mathematical approach based on set theory (Briand et al., 1996) to the identification of a set of nine abstract properties to which a complexity metric must adhere (Weyuker, 1988). The seminal leaders in the field of software validation all take the position that a measure or metric needs to go through a rigorous validation process to ensure the measure or metric is appropriate for the measurand. However the specific process put forth is based on the individual author’s proposed framework (Schneidewind, 1992; Kitchenham et al., 1995; Briand, and Melo, 1996). Because validation efforts use a researcher’s validation framework, a measure can pass one set of validation criteria yet fail another. The differences among the focus of specific research efforts, as shown in Table 12, continue today (Meneely et al., 2012).

As previously discussed, there are different interpretations of complexity. Using set theory, Weyuker (1988) evaluated complexity based on nine proposed properties that a measure should exhibit to be considered validated. Weyuker described each of the nine properties textually as well as mathematically. The description defined the essence of the

Table 12

Divergent Foci of Software Metric Validation Frameworks

Researcher focus		
Researchers	Year	Focus
Weyuker	1988	Properties required for complexity metric.
Fenton, Kitchenham	1990	Predictability, applicability to measurand, scalability
Schneidewind	1992	Definitions an framework culminating in IEEE Standard 1061-1998 IEEE Standard for a Software Quality Metrics Methodology
Kitchenham, Pfleeger, & Basili	1995	Inferential statistical support for validation of Chidamber & Kemmerer (1994) metrics.
Briand, Morasca, & Basili	1996	Validation based on mathematical set theory.
Basili, Briand, & Melo	1996	Inferential statistical support for validation of Chidamber & Kemmerer (1994) metrics.
Meneely, Smith, & Williams	2012	Meta-analysis leading to 47 validation criteria.

property and the corresponding set notation defined the mathematical property associated with the description. The nine properties, as described by Weyuker (1988), follow. The word *program* is used to mean any *program body*. The following notation is used to represent the essence of the property in set notation: $c(A)$ represents the complexity of a program A, $\forall$ represents for all, f represents functionality, and $\exists$ means exists.

1. Any measure that measures all measurands with the same complexity is not a measure.

   $$(\exists A)\ \&\ (\exists B) \text{ such that } c(A) \neq c(B) \quad (1)$$

2. A measure must be sensitive enough to divide complexities into a finite, but not too coarse, set of levels of complexity.

   $$\text{For } n > 0,\ (\exists A) \text{ such that } c(A) = n \quad (2)$$

3. Two programs can have the same measure.

$(\exists A)\ \&\ (\exists B)$ such that $c(A) = c(B)$ (3)

4. Programs with the same functionality can have different complexity.

$(\exists A)\ \&\ (\exists B)$ such that $((f[A] = f[B])\ \&\ (c[A] \neq c[B])$ (4)

5. When concatenating the $\exists$ complexity of two programs, the complexity of the resulting program is greater than either of the two programs individually.

$(\forall A)\ \&\ (\forall B),\ (c[A] \leq c[A + B])\ \&\ (c[B] \leq c[A + B])$ (5)

6. When two programs of equal complexity are concatenated individually with a third program the complexities of the two individual concatenations are not equal.

$(\exists A)\ \&\ (\exists B)\ \&\ (\exists C),\ (c[A] = c[B])\ \&\ c(A + C) \neq c(B + C)$ (6)

7. It is possible for a change in the order of operations of a program to change the complexity.

If B represents a change in the order of operations of A, $c(A) \neq c(B)$ (7)

8. Renaming a program will not change the complexity,

If B represents a renaming of A, $c(A) = c(B)$ (8)

9. Adding components to a program can increase the complexity.

$(\exists A)\ \&\ (\exists B),\ (c[A] + c[B]) < c(A+B)$ (9)

McCabe's CCM was previously introduced as a continuing measure of algorithmic complexity. McCabe's CCM did not satisfy Weyuker's 2nd Property, described in Equation 2 (Weyuker, 1988). The essence of Weyuker's 2nd Property is to

ensure that a measure is sensitive enough to measure the measurand. McCabe's CCM allows multiple programs that have a wide variation in functionality, yet the same computational paths, to have the same complexity and therefore is not sensitive enough. Stated mathematically, it is possible for the following to occur. Given that programs A, B, and C exhibit a wide variation in functionality yet have the same computational path, thus the same complexity, then

$$([\exists A] \,\&\, [\exists B] \,\&\, [\exists C]) \,\&\, (f[A] \neq f[B] \neq f[C]) \,\&\, (c[A] = c\,[B] = c[C]) \quad (10)$$

Weyuker's analysis continued and documented that Halstead's measure of effort failed Weyuker's 5th Property.

Because multiple interpretations of complexity exist, the validation process utilized by Briand et al. (1996), also based on set theory, would incur different results depending upon the specific interpretation of complexity upon which the measure is based (Poels & Dedene, 1997). Briand et al. confirmed that their set theory approach is "convenient and intuitive" (p. 68). Poels and Dedene (1997) commented further by indicating that the properties identified by Briand et al. were necessary but not sufficient and continued to add that the properties were appropriate for invalidating a measure but not sufficient for validating a measure.

Using meta-analysis, Meneely et al. (2012) concluded that (a) metrics validation was not simple, (b) multiple motives and philosophies existed behind the identification and development of metrics validation mechanisms, and (c) current validation approaches

represent a researcher's perspective and opinion. Metrics validation is at a similar level to CMMI Level 1 for software development maturity—ad hoc.

Reaching agreement on metrics validation is elusive and made more difficult when there are inconsistencies among the definitions of the measures and metrics to be validated. Without agreement within the community, managers of software development efforts must rely on what a specific project needs at a specific point in time given project specific resource constraints. Faced with lack of clarity, corporate management must make a decision with respect to initiating and/or implementing a metrics improvement program.

Technical reports, specifically CMU/SEI-93-TR024, CMU/SEI-93-TR025, and CMU/SEI-92-TR020 (Park, 1992, Paulk et al., 1993, 1994) and Standards, specifically IEEE 982.1™-2005, IEEE 1061-1998, ISO/IEC 15939, ISO/IEC/IEEE 24765 contain definitions of, and frameworks for, metrics collection and analysis programs (JTC1SC7, 2007; JTC1SSESC, 2010; SESC, 2006, 2009). Again, there are inconsistencies among the report and standards. They do not agree. One report, specifically ISO/IEC/IEEE 24765, provides three different definitions of measure within the document: first a measure is a variable, second a measure is a comparison of a measure with a baseline value, and third a measure is the action of applying a measure to an attribute (JTC1SSESC, 2010). The software engineering discipline is an engineering discipline yet does not exhibit the degree of scientific rigor necessary to address metrics and metrics validation.

Summary: Software metrics validation. A myriad of measures have been identified, many with ambiguous definitions that result in lack of industry agreement. These various interpretations of measures, at the very least, make validation difficult, especially when multiple validation frameworks exist that differ in the approach to validation. One measure may be validated in one validation approach and not validated in another validation approach. The state of validation of measures is identical to the state of measures. There are inconsistencies in both what the measures mean and how to validate measures. This leads to the question: *How can a measure, ill defined, be validated by a framework that has not achieved agreement in the industry?*

These measures, whether validated or not to one of many differing validation frameworks, are then used as input to one of many project prediction/estimation software packages that produce differing results when estimating software size, cost, schedule, and level of effort. The estimates that result from using estimation software packages are estimates, based on estimates, that are then used as the basis for negotiations on multi-million dollar contracts.

Research Approach

Given the following conditions, the (a) continuum of management mechanisms, (b) inability of management truisms to detect completion status, (c) inability of software measures and metrics to detect completion status, and (d) the magnitude of software project overruns, an hermeneutical phenomenological research effort focused on obtaining and analyzing management experiences could begin to bridge the gap between

existing and ongoing research and managements' needs. Management truisms are just that, truisms; generalities and guidelines that are applicable but not necessarily implementable due to budget, schedule, and resource constraints. The metrics available to management are questionable. When applied identically, and/or a history of usage exists, metrics can be supportive but not for detecting development completion status. My research effort focused on gathering managements' experiences with metrics as a first step towards understanding managements' needs with respect to completion status.

Phenomenological Research Approach

Phenomenological research is applicable for eliciting managements' experiences with the phenomenon of management metrics and relevancy to incremental software development completion status. The word hermeneutic derives from the Greek word, *hermeneuo*, and means to interpret. Applied to phenomenology, the term hermeneutic phenomenology means to interpret the lived experiences of the participants, whether from textual descriptions (Moustakas, 1994), artifacts, or observations (Patton, 2002). The choice of phenomenological research is grounded on the difference between the words explain and explore. This study does not look to explain the relationships between the two variables metrics and relevance to software status, but rather seeks to explore managers' lived experiences with current metrics and the relevance of those metrics to assessing and reporting software completion status.

Phenomenological Qualitative Research

To address the research questions, a phenomenological approach best supported the goal of (a) exploring the lived experiences of stakeholders with respect to software metrics and detecting/reporting software completion status, and (b) providing a fresh perspective to the phenomenon of detecting and reporting software completion status from management's perspective. Phenomenology has been used in multiple and diverse industries, including but not limited to the nursing profession to explore the needs of patients (Cohen, Kahn, & Steeves, 2000), the information technology community to discover the experiences of managers when dealing with unstructured data on servers (Tigari, 2012), and in the military community to determine the task-technology fit of simulation training in a military environment (Cane, McCarty, & Halawi, 2009).

First, within the nursing profession, exploring the perceived needs of patients to better meet their needs is a classic example of phenomenological research (Cohen et al., 2000). The goal was to better understand the effect of disease on patient's lives. Focus was not on dealing with the illness or experiences with treatment, but rather on the effect the illness had on patients lives. Second, phenomenological research was conducted to explore the self-perceived benefits of simulation training on task to technology fit within the military (Cane et al., 2009). Previous quantitative analyses, conducted on task to technology fit and self-perceived improvement in management performance, concluded with diverse results. This study extended those quantitative studies by conducting a qualitative phenomenological study. The results added themes to review within the

context of the previous quantitative results findings, similar to a mixed methods study conducted by two different researchers. A third effort utilizing phenomenological research was focused on exploring and describing the experiences of Information Technology (IT) professionals responsible for decisions with respect to the management of unstructured data stored on a network (Tigari, 2012). Tigari identified sixteen themes to support management with managing data stored on a network, for example management strategies, staff training, security, and the necessity to manage unstructured data.

Sampling Strategy

Two concepts in common to all qualitative inquiry, actually all research in general, are sampling strategy and sample size. Qualitative research methods support multiple sampling strategies and require a balance of breadth and depth of participants in the sampling strategy. The sample must be broad enough to allow growth of concepts and categories of data yet provide sufficient detail to support the capture of relevant data (Patton, 2002). Stated differently, when a sample size is too large, the details may be lost and when a sample size is too small, it may be difficult to support the research goals (Sandelowski, 1995). Whether using quantitative or qualitative analysis, sampling strategy and sample size are both critical to the acquisition of quality data which in turn leads to a quality research effort. Balance in this research was supported by a sampling strategy that clearly identified the criteria for participation as well as the number of participants. Creswell (2007) identified phenomenological studies with sample sizes

ranging from 1 <= n <= 25.

A researcher can choose a single sampling strategy from among those provided by Nachmias and Nachmias (2008) or a mixture of multiple sampling strategies (Creswell, 2007). Often a mixture of strategies provides flexibility in obtaining a sample by triangulating various concerns such as cost, time, and sample size requirements of breadth and depth.

Qualitative Phenomenological Research Summary

Phenomenological research is an attempt to objectify what is subjective, specifically researchers' synthesis of participants' lived experiences, as illustrated in Figure 6. Participants share their lived experiences of a phenomenon with a researcher who then synthesizes, and analyzes the descriptions of the experiences, and concludes by identifying theoretical constructs, or key findings, and documenting the path taken to arrive at those findings.

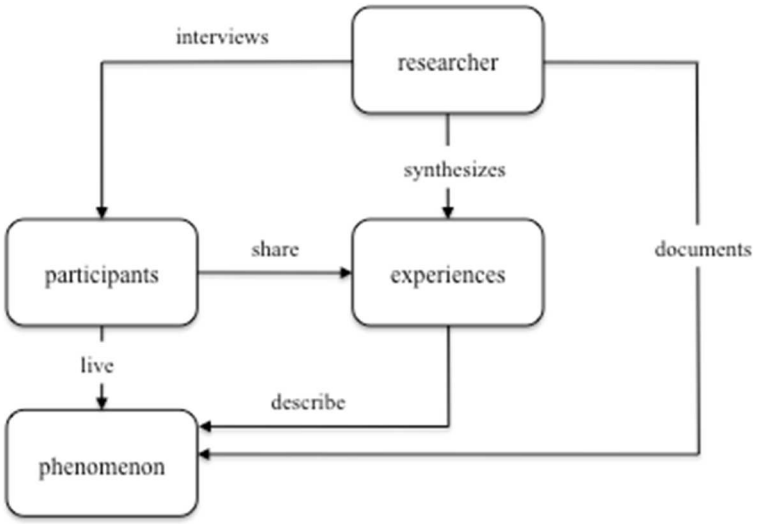


Figure 6. Phenomenological research overview (figure created by P. Texel using Microsoft PowerPoint).

Despite variations in approaches to phenomenological research, there are concepts that are intrinsic to all the approaches. These core concepts include, but are not limited to, intentionality and descriptions of the essences of experiences. Intentionality is related to the concept of human consciousness opening up to the phenomenon, allowing the phenomenon to be fully lived and described. Participant descriptions of lived experiences are the source of research data and the cornerstone of a phenomenological research study.

Summary and Conclusions

I identified a gap in the literature with respect to mechanisms that provide management insight into software development completion status. Multiple reports documented national and international cost and schedule overruns for both commercial and defense software applications. Lack of management insight was repeatedly cited as one, of many, issues responsible for these cost and schedule overruns that continue to plague the software industry.

Existing management mechanisms lie on a continuum that at one end provides abstract management guidelines and technical syntactic metrics on the opposite end. Management guidelines are truisms based on years of experience provided to support and guide overall management but these truisms do not address software completion status. Technical syntactic metrics are based on program code and (a) lack consistency with respect to their very definition, (b) lack rigorous scientific validation, and (c) address

characteristics of software other than completion status (e.g., complexity, quality, level of effort, and size).

A review of the literature indicated that management does not have the necessary mechanisms to provide insight into completion status. Despite the lack of relevant data, management must still report completion status to all stakeholders. However, without relevant data, reporting status often places management in a compromising position. A qualitative hermeneutical phenomenological research effort was proposed to explore management's positive and negative experiences assessing and reporting completion status with current support data.

The importance of gaining a fresh perspective from those who are experiencing and living a phenomenon cannot be understated, even when the perspective gained is either similar to or different than that expected by the researcher. The specific framework, process, and detailed steps describing the proposed approach to conducting this research are elaborated in Chapter 3.

Chapter 3: Methodology

The literature review supported the assertion that multiple management mechanisms exist to measure software products. These measures exist on a continuum: abstract management guidelines on one end, and detailed technical measures extracted from program code on the other end. The problem is that current mechanisms do not measure software completion status but rather other characteristics of software products (e.g. risk, size, complexity). My research study consisted of exploring program and project managers' experiences of assessing and reporting software completion status when measurements of completion status were lacking. The research questions, previously identified in Table 2, are repeated here for convenience as Table 13.

Table 13

Research Questions

	Research questions
RQ1	How have current software metrics supported the assessment of software development completion status?
RQ2	How have current software metrics supported the reporting of software development completion status?
RQ3	What is the relevancy of software metrics to Software Development Life Cycle (SDLC) phase?

A phenomenological research study was an appropriate choice to explore experiences of detecting and reporting software development completion status. As discussed in Chapter 2, phenomenology, in its simplest form, is the study of phenomena, with a *phenomenon* defined as an entity (e.g., event, object) that exists in reality and

within the context of the mind. The opposite of phenomenon is *noumenon*, an object independent of the context of the mind and without any context: the entity itself devoid of any context. A noumenon is an entity; a phenomenon is the experience with that entity (Cohen et al., 2000). In drawing a parallel to my research, one could say that program and project managers' experiences with software metrics are distinct from the software metrics themselves.

My research focused on the phenomenon, as lived by program and project managers, of experiences with the assessment and reporting of software development completion status of software-intensive systems using current software metrics. Synthesizing and sharing these experiences would provide a common ground for both researchers and practitioners with respect to software community needs to effectively support the assessment and reporting of incremental software completion status.

As previously specified in Chapter 2, I chose to use hermeneutical phenomenology for my research. Hermeneutical phenomenology focuses on interpreting response data captured from participants' interviews, from both an internal horizon (the researcher's interpretation (of response data) and an external horizon (the factors that contributed to the experience; Moustakas, 1994). Operationalized to my research, the dynamics of government contracting provided the external structure, the external horizon, within which contractor software development occurs. After a justification of the research design and rationale, subsequent subsections include a discussion of the specific operationalization of the methodology to this research and the issues related to the four

components of trustworthiness: transferability, credibility, dependability, and confirmability (Shenton, 2004).

Research Design and Rationale

The following subsections provide a justification for why a qualitative approach was selected over a quantitative or mixed-method approach, why a phenomenological approach to qualitative research was chosen over four alternative qualitative approaches (e.g., ethnology, narrative), and why an hermeneutical phenomenological approach was chosen over alternative phenomenological approaches (e.g., eidectic, relational).

Justification for Qualitative Approach

There are three core approaches for a research study: quantitative, qualitative, and mixed methods. The first approach, quantitative, is deductive in nature, beginning with a theory, followed by testing to confirm, or not, the hypotheses identified for a study. Descriptive and inferential statistics are major components of quantitative analysis, relying heavily on the identification of dependent and independent variables and the relationship between them. My research was not focused on the relationship between software metrics and the assessment and reporting of software completion status, but rather managers' experiences living with software metrics as a mechanism for assessing and reporting software completion status. Quantitative analysis was not applicable to this study.

The second approach, qualitative analysis, was applicable to this study. A qualitative analysis approach, inductive in nature, begins with observations, builds

patterns from those observations, and ultimately generates theoretical constructs based on those patterns that are not generalizable. The word *observation* is overloaded in the research discipline and can mean either direct study of participants, as in watching participants while a researcher is embedded with participants in the field, or any data collected by a researcher that are useful for a research study: field observations, documents, audio tapes, video clips and so on.

Third, a mixed methods approach is a combination of quantitative and qualitative approaches and would be applicable, specifically sequential mixed-methods (qualitative followed by quantitative); however, mixed-methods approaches are resource intensive and thus difficult to complete in a timely fashion. Future research could follow this qualitative study with a survey-based quantitative study to examine the relationship between the two variables: software metrics (independent variable) and completion status (dependent variable).

Justification for Phenomenological Qualitative Approach

Given that a qualitative approach was the chosen methodology, there were five qualitative research approaches from which to choose. The identification and focus of the five qualitative research analysis approaches are listed in Table 14. My research was not focused on one participant's chronological experiences with software metrics (narrative), nor was there sufficient literature on the relevance of software metrics and software completion status from which to develop a theory (grounded theory), nor could participants be released from assignments in the workplace to participate in a case study;

finally, the culture of managers of software-intensive development efforts would not have addressed the research questions (ethnography). When I considered the focus of each of the five approaches, the choice applicable to the research questions of this study was clear: phenomenology.

Table 14

Comparison of Focus: Five Qualitative Approaches

Focus of five qualitative research approaches	
Research type	Focus
Narrative	A story, potentially chronological, of a single individual, with respect to a topic
Grounded theory	A theory evolved and underpinned by literature
Case study	Bounded study of an event
Ethnography	Exploration and study of a culture
Phenomenology	Exploration of lived experiences

Justification for Hermeneutical Phenomenological Approach

Approaches to objectifying subjective data differ due to a researcher's underlying philosophical alignment; consequently, phenomenological research has multiple operationalizations based on a philosophical lens. For example, eidectic reduction is based on a researcher introducing purposeful variations of the phenomenon to gather common themes while ignoring differences based on the variations. An additional operationalization of phenomenological research, relational research, places emphasis on the researcher's approach and the data mining process used to analyze research response data. A hermeneutic approach to phenomenology, based on interpretations of described

experiences, is distinct from a heuristic approach (Moustakas, 1994). A hermeneutic approach represents the breadth of an experience, including interpretation of experiences and the environment within which the experience is encountered, while a heuristic approach represents an exploration of the depth of the experience, including multiple artifacts related to the experience, such as additional texts, video, or music (Moustakas, 1994).

Role of the Researcher

In quantitative research, statistical tests are the data analysis mechanisms. The researcher is the data analysis mechanism in qualitative research. Because of the criticality of the researcher in qualitative research, the experience of the researcher related to the study topic, along with any biases, must be made known to participants and be included in any research documentation. Researcher experience with the research topic adds to the credibility of the research analysis and results.

I managed my bias, based on 25 years of experience with software development projects, by maintaining a focus on the transcriptions of the interviews and the clear guidelines for coding as presented by Saldaña (2013) and Auerbach and Silverstein (2003). Saldaña placed emphasis on the detection of coding methods in an initial coding cycle (e.g., grammatical, elemental) augmented by examination of the transcription with respect to descriptive, emotion, and other patterns of data that encompass various words in the transcription. Auerbach and Silverstein emphasized focus on really listening (with eyes that examine the text) to what the participants shared. A qualitative analysis

researcher must approach the analysis of qualitative data scientifically and methodically to increase the credibility and transferability of the results. I was the only coder and followed open and axial coding as proposed by Saldaña's coding methods. Simultaneously I maintained listening eyes, a strategy put forth by Auerbach and Silverstein. These two methods provided a significant contribution to intra-coder reliability.

Methodology

The operationalization of the hermeneutical phenomenological theory of research introduced in Chapter 2 incorporated a convenience, purposeful, and snowball sampling strategy to extract a representative sample from the accessible population. All members of the sample participated in an interview focused on obtaining experiential data that was then analyzed for conceptual themes that synthesize the participants' experiences. The details of the research process, from sample selection to data analysis, are summarized in four major subsections: Sampling strategy (population, sampling frame, sample), pilot study (questionnaire, interview), data capture (questionnaire, interviews), and data analysis (descriptive statistics, thematic analysis). The specific identification of each individual step in the process is contained in Appendix H. The content of Appendix H represents the data provided as part of the IRB Application. Chapter 4 contains descriptions of modifications, made in response to IRB comments.

Sampling Strategy and Sample Selection

The theoretical population consists of program and project managers managing software-intensive government agency applications. The accessible population was the set of four community partner members of the population who had developed, or were currently developing, software-intensive systems monitored by the contracting government agency. The point of contact (POC) for each community partner provided the sampling frame according to participant selection criteria. A purposeful criteria-based analysis of the sampling frame resulted in a sample of 24 participants. The sample size, 24, was consistent with the guidelines established by Creswell (2007). The selection criteria for participants follow:

- Minimum age of 25 years
- Minimum of 2 years of experience managing/ monitoring software-intensive military applications
- Minimum of 2 years of experience reporting/monitoring software completion status for software-intensive military applications internally within the organization
- Minimum of 2 years of experience reporting/monitoring software completion status for software-intensive military applications externally to stakeholders

A mixed sampling strategy, satisfying Creswell's (2007) guidelines for convenience, criterion, purposeful, and snowball sampling (discussed in Chapter 2) and illustrated in Figure 7, guaranteed that all participants had experience using software

metrics as both an assessment and reporting mechanism for software completion status. The final sample size was 20. There were four additional participants in the initial sample who disengaged from the study.

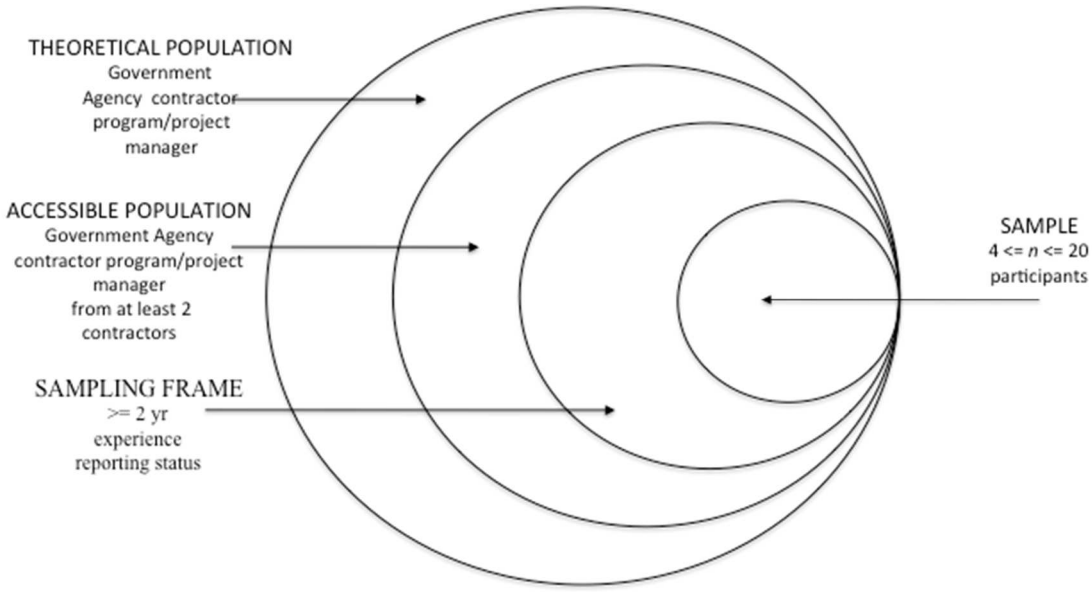


Figure 7. Sampling strategy (figure created by P. Texel using Microsoft PowerPoint).

Sampling process tasks. The following activities led to the identification of the sample:

1. Contact organizations. Researcher contacted POC in each organization by telephone, provided a copy of the Proposal and blank Letter of Cooperation to POC (see Appendix C), obtained a signed Letter of Cooperation from the POC, provided POC with criteria for inclusion in sampling frame, and requested a sampling frame.
2. Generate sampling frame. POC generated the sampling frame based on the criteria provided and distributed sampling frame to me.

3. Create sample. I created a purposeful sample and provided a list of potential participants to POC as well as Consent Form (see Appendix D) for distribution to potential participants. The Consent Form contained no geographical or organizational identifying information.
4. Contact potential participants. POC contacted each potential participant and introduced Consent Form (see Appendix D) and Invitation letter (see Appendix E).
5. Identify sample. Participant, if in agreement, returned a completed Consent Form as a password-protected Word document to me via email with the words "I agree." I printed the Consent Form and email and stored the hardcopy in a locked container. All emails were also maintained as a password-protected file.
6. Create Participant Mapping Table. I assigned each participant an ID number as defined in Table 15.

Table 15

Participant ID Mapping Schema

Participant ID mapping schema	
Pilot study participant number	Pilot study participant name
P*i*	< Participant name >
P*j*	< Participant name >
P*k*	< Participant name >

Note. P*n,* 1 <= *n* <= 20was used to identify participants in the study. Pilot study participant numbering is not sequential due to the random nature of selecting participants.

At the conclusion of sample selection, four community partners provided a sample of 20 participants. The use of either password-protected Word files, hardcopy stored in a locked container, or both, protected participants' privacy. Apple's Time Capsule and Time Machine software automatically backed up project data. Additionally, I manually backed up data on a USB flash drive that served as both a backup device and final storage media for research electronic artifacts.

Conduct Pilot Study

In addition to ensuring the collection of the required response data, I conducted a pilot study to discover any anomalies in the research process that could be corrected before the main research study began. The purpose of the pilot study was to ensure that the questionnaire focused on collecting demographic data and that semistructured interview questions focused on eliciting experiences supporting the capture of the desired research data. Prior to providing the detailed steps of the pilot study process, there was a common thread throughout the entire research process—the use of a third-party software application, NoNotes.com.

NoNotes.com. NoNotes.com is an iPhone application downloaded from the NoNotes.com web site (http://www.NoNotes.com). The subscriber, using either the NoNotes.com application on an iPhone or a landline, places a telephone call to the participant. NoNotes.com records and transcribes the interview. The transcription, emailed as plain text to the NoNotes.com subscriber (the researcher), identifies the participants in the conversation as Speaker 1 and Speaker 2. The transcription does not

reference the names of the two members of the conversation. Additionally the transcript does not reference either the researcher or participant's telephone number. Appendix A contains an excerpt from a transcription generated using NoNotes.com. Once NoNotes.com forwards the transcription file to the subscriber, the subscriber (the researcher) deletes the audio and transcription files from the NoNotes.com server.

I interviewed participants in both the pilot study and the main research effort using NoNotes.com. I initiated each telephone call using the NoNotes.com app on my iPhone. Analysis of the pilot study transcriptions did not The textual transcriptions of the conversations that took place during the pilot study were not analyzed using NVivo for the pilot study but did become input to NVivo for the data analysis component of the main research study.

The pilot study process. For each participant I generated a concept map based on participant response data. Analysis of the concept maps indicated whether the questionnaire and interview protocol needed refinement to ensure the relevancy of the responses to the research questions. The following list identifies the steps performed to conduct the pilot study, in the order in which they were performed.

1. Select pilot study participants. I selected a random sample of three participants from the sample and contacted each participant by email to establish a mutually agreed interview date and time.
2. Distribute interview instruments. I distributed a copy via email of the questionnaire and interview protocol to pilot study participants at least

24 hours in advance of the established interview date and time.

3. Conduct interview. I conducted the interview using NoNotes.com. During the interview, I annotated my hard copy of the questionnaire and interview protocol as required and stored the annotated hard copy in a locked container. The annotated hard copy included additional researcher comments related to the topic of the question as well as any interpretation of participant mood (e.g., seemed anxious, cares about the issue, or seemed annoyed and preoccupied).
4. Receive transcription. NoNotes.com forwarded the unedited transcription of the interview to me and I migrated the transcription file (.txt) to (a) a password-protected .zip file to maintain an audit trail, (b) a password-protected Word file (.docx) for exchange with the participant, and (c) deleted the original file from the NoNotes.com server.
5. Conduct member-check**.** An email exchange of the password-protected Word file representing the original transcription enabled the member check process. The member check process continued until a participant indicated agreement with the content of the transcript.
6. Conduct initial analysis. I conducted an initial analysis of responses for key concepts using visual inspection and ensured the relevancy of key concepts to the research questions. I created the concept maps,

from the interview content, to capture concepts and their relationships. Further analysis of pilot study response data occurred during the analysis of all response data from all participants.

7. Update interview instruments. I updated the questionnaire and interview protocol as needed.

Pilot study completion status. At the conclusion of the pilot study I deleted initial interview transcripts from the NoNotes.com server, stored member-checked data of participants' transcriptions as sanitized Word files, and updated the questionnaire and interview protocol to support migration to the final questionnaire and final interview protocol. The content captured in the initial concept maps led to the establishment of the initial NVivo node structure; concepts only, no coding. Incorporation of pilot study demographic and experiential data took place during data analysis of the main research study. I added all pilot study artifacts to the USB flash drive initiated during sample selection.

Data Capture

Upon completion of the pilot study and finalization of the sample, the data capture component of the main research study began. This phase of the study focused on (a) obtaining the remaining participants' responses to the questionnaire and interview protocol questions in a sanitized format, and (b) maintaining the initial NVivo project by augmenting the initial node structure, and (c) importing sanitized raw data into the project. NVivo maintains textual documents, called Sources, as part of a project

database. At this point in the project, I imported all sanitized transcriptions, as Word documents, into the NVivo database using the import sources functionality. The process of capturing data did not include data analysis, but initiated an NVivo project that included importing the sanitized Word documents into the NVivo project using the NVivo import functionality. A subsequent subsection, titled Data Analysis, contains a discussion of the process used to analyze the project's response data.

Questionnaire. The questions in the questionnaire content captured participant demographic data. The focus of each question in the questionnaire is listed in Table 16. The questionnaire, a Word form, is included as Appendix F. The questionnaire in Appendix F represents modifications (identified in Chapter 4) that were approved by the IRB. In addition to gathering demographic data, the questionnaire served a second purpose, to ensure the participants conformed to the selection criteria.

Table 16

Questionnaire Content

Questionnaire content	
#	Question focus
1	Years of experience
2	Current involvement with government contract
3	Existence of IV&V involvement
4	Type of government agency (DOD or non-DOD)
5	Current role
6	CMMI level
7	Report internally
8	Report externally

Interview protocol. The interview protocol, included in Appendix G Figure G1, consisted of eight semistructured questions designed to probe the participants' memory of experiences with software metrics and the assessment and reporting of software completion status. The questions focused on (a) positive experiences with assessing software completion status, (b) challenges with assessing software completion status with current metrics, (c) positive experiences with reporting software completion status, internally within the organization and externally to stakeholders, (d) challenges with reporting software completion status, internally within the organization and externally to stakeholders, and (e) the relevance of existing metrics to SDLC phases. Lastly, the design of the interview questions focused on the coverage needed to support the research questions (see Appendix G Table G1).

Data capture process. The steps that were performed, in the order in which they were performed, to conduct the data capture component of this research study follow:

1. Conduct interview**.** For each participant, the researcher (a) established interview date & time for each participant, (b) distributed questionnaire and interview protocol to participant at least 24 hours in advance of the established interview date and time, and (c) interviewed participant using NoNotes.com.
2. Annotate Interview Protocol. I annotated hard copy of the questionnaire and interview protocol and stored the hard copies in a locked container.
3. Receive transcription. NoNotes.com forwarded the unedited transcription of the interview to me as a .txt file and I (a) migrated the transcription file (.txt)

to (a) a password-protected WinZip file (.zip) to protect participant confidentiality, (b) maintained an audit trail, (b) password-protected the Word file (.docx) for exchange with participant, and (d) deleted the original file from the NoNotes.com server. All files exchanged with participant are now password-protected

4. Member-check. The password-protected Word file representing the original unsanitized transcription was exchanged between the participant and the researcher until final agreement was reached with respect to the content.
5. Participant approval. I transmitted the final sanitized file to a participant and awaited a response from each participant indicating "I approve" before analysis could begin.
6. Participant privacy. I ensured the Word file was sanitized, that there was no identifying information to identify a participant, organization, or geographic location within the contents of the file to import into NVivo. This file is not password-protected.
7. Initial visual analysis. I analyzed responses for key concepts using visual inspection of responses, developed informal concept maps, and utilized the findings as the basis to augment the initial node structure in NVivo. I imported the sanitized Word files representing participant response data into NVivo.

Data capture summary. The following represent the completed research

products at the conclusion of data capture: (a) audit trail of password-protected unsanitized transcription file; (b) initial NVivo database (specifically initial node structure and sanitized response data imported to the database) in preparation for data analysis; (c) original unsanitized transcription files deleted from the NoNotes.com server; and (d) all password-protected files, Word (.doc) and text (.txt), as well as all sanitized files, added to the USB flash drive initiated during sample selection.

Data Analysis

Two mechanisms generated analysis of the response data: descriptive statistics summarized the demographic data captured by the questionnaire and qualitative analysis conducted on the content of the experiential data, captured by interviews, led to the key findings discussed in Chapter 4.

Obtain descriptive statistics. I entered the demographic data, captured by the questionnaire, into IBM SPSS to obtain the descriptive statistics, specifically frequency distribution of the participants and basic central tendencies, for example years of experience. Descriptive statistics included frequency counts and percentages related to CMMI level, role, and gender. Central tendency statistics for the years of experience included mean, standard deviation, range, median, mode, and quartile percentages. Descriptive statistics also supported the identification of potential outliers in a sample, specifically those participants that do not satisfy the selection criteria. Based on researcher judgment, there were no outliers.

Conduct qualitative analysis. NVivo, a software tool that supports qualitative analysis, (a) maintained sanitized response data, (b) supported the documentation of research data content analysis, (c) supported the cognitive process of thematic analysis, and (d) provided selected graphical representations of emerging patterns (e.g., cluster map, concept map).

Using NVivo's three basic activities (specifically coding, categorization, and thematic analysis), I migrated raw data to the theoretical constructs that address the research questions. Coding focused on the decomposition of response data into words and phrases (codes) that represent the concepts, patterns, and relationships between concepts. Following decomposition, codes are then reassembled into categories based on similarities. Continued examination and analysis of the identified codes and categories led to emerging themes.

NVivo, well suited to the documentation of a researcher's progression through this migration of raw data to themes, maintained and supported the analysis of qualitative raw data. Specifically NVivo supported (a) nodes and subnodes to organize raw data into categories, and (b) codes, the allocation of a fragment of text to a node (Bazeley & Jackson, 2013).

NVivo supports a flexible and powerful search capability (Bazeley & Jackson, 2013). A search can be limited to an exact word, for example *discomfort*, or can be widened using a slider bar to permit similar words to be included in the search, for example *uneasy* (Bazeley & Jackson, 2013). Dependent upon the search filter used, the

NVivo display presents a list of text from the sources in the database that fulfill the search criteria, enabling the detection of repeated words or phrases across multiple input sources (Bazeley & Jackson, 2013).

Lastly, NVivo can generate a model, similar to and approximating a concept map (Wheeldon & Ahlberg, 2012). An NVivo model is a graphical representation of nodes, sub-nodes, and relationships between them that represent the main concepts in a study and the identified relationships between them (Bazeley & Jackson, 2013).

Migration of raw data into thematic constructs is a highly conceptual process and an extremely iterative process. As illustrated in Figure 8, nodes and codes are continually refined and continue to evolve as the cognitive process unfolds. A newly added node may impact previous codings and new codings may impact existing node structure.

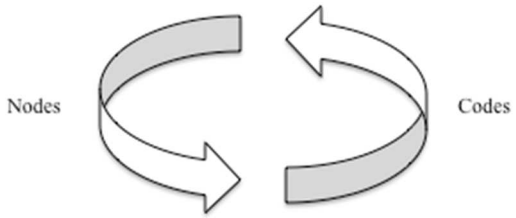


Figure 8. Iterative process of node and code identification (figure created by P. Texel using Microsoft PowerPoint).

The substeps that supported the basic three steps previously identified are specified below:

Initial read of response data. Qualitative data analysis began by first reading all transcriptions of participants' interviews. This process is similar to a literature review. A researcher is surrounded by different views of concepts that must be synthesized and analyzed (Saldaña, 2013).

Develop concept maps. For each participant, I developed an informal paper and pencil concept map, hand drawn to initially support the identification of abstract concepts represented in a participant's transcribed interview. I later reproduced and refined the concept map as a Microsoft Power Point file. A concept map identifies common concepts, as well as variations, within response data to assist in the modification of the initial NVivo node structure previously established in the pilot study. The development of concept maps was the initial step in migrating the initial raw response data representing participants' experiences to a final analysis report that provided content for Chapters 4 and 5.

Update initial NVivo node structure. I updated the initial node structure, if necessary, as a result of initial review of the response data. Updates included addition of new nodes/subnodes as necessary. Additionally, if two nodes represented the same concept, yet had different nomenclature, I then combined the nodes into one node.

Iterate to migrate raw data to results data. Following the first reading of the interview transcriptions and the establishment of the updated node structure, I followed the iterative process between and among node/code modifications, illustrated in Figure 8, until the following occurred: (a) stability of the nodes, meaning no new nodes were conceived; and (b) the stability of the codes, meaning all relevant text had been coded to a node or nodes.

I migrated raw data to theoretical constructs following a process based on the following cognitive guidelines: (a) filter out irrelevant data, (b) examine relevant data for

repeated ideas (found in the use of similar words), (c) group repeated ideas into themes, and (d) abstract themes to a theoretical construct that directly addresses the research questions (Auerbach & Silverstein, 2003). As depicted in Figure 9, these cognitive guidelines form a pyramid with raw research data forming the base of the pyramid, each layer of the pyramid building on the previous layer, and ultimately concluding with a set of theoretical constructs that address the research questions. The iterative process continued until the termination criteria, previously stated, had been met.

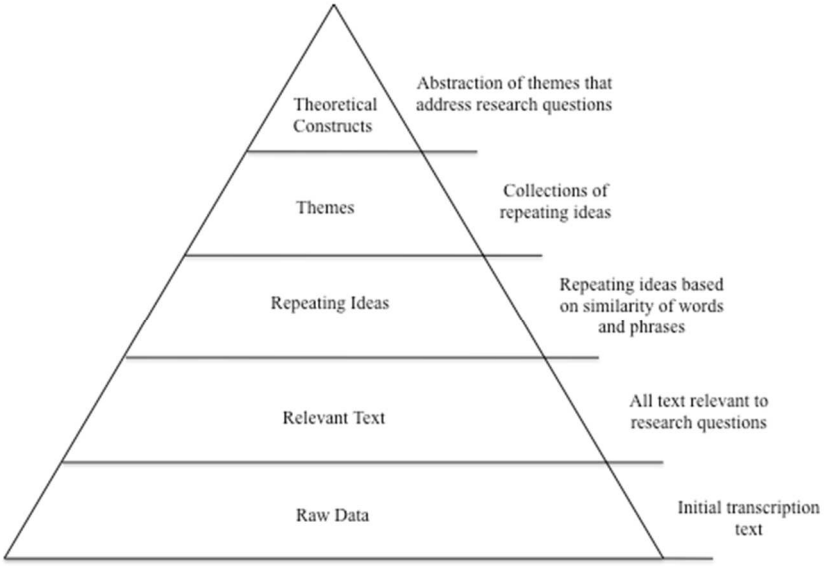


Figure 9. The qualitative coding process (figure created by P. Texel using Microsoft PowerPoint 2011).

The philosophy underpinning the iterative migration process combined both open and axial coding (Creswell, 2007) but under the umbrella of Saldaña's (2013) process. Open coding breaks down textual data to raw elements while axial coding reassembles

the raw elements into categories based on similarities and differences. Saldaña's (2013) process of *first cycle* coding methods and *second cycle* coding methods is one operationalization of open and axial coding.

First cycle methods define criteria to support the extraction of atomic elements from raw text. I did not use all of Saldaña's criteria for first cycle coding. Data analysis began with grammatical methods (e.g., attribute, magnitude, and simultaneous coding), *elemental methods* (e.g., structural, descriptive, and initial or open coding), *exploratory methods* (e.g., holistic), and *affective methods* (e.g., emotion and evaluation coding). Additional first cycle coding methods may be employed depending upon the current state of the coding analysis. However, I took care to not use *contradictory* coding methods (Saldaña, 2013). For example mixing *exploratory* and *procedural* coding methods would provide two divergent views into a study that are not appropriate. This research focused on experiences reporting software completion status, not the process of obtaining report data.

Second cycle coding methods aggregate, or reassemble, first cycle coding results into more abstract patterns and classifications through researcher conceptualization (Saldaña, 2013). Specific second cycle coding methods that I employed were *pattern* (e.g., commonality in initial codes), *axial* (e.g., reassembling low level data elements from first cycle coding), and theoretical (e.g., collection of categories into themes).

Generate NVivo reports and graphs. I generated NVivo final reports and graphs in preparation for inclusion in Chapter 4, Results.

A completed NVivo model, thematic constructs that addressed the research questions, and NVivo project data were added to the USB flash drive and marked the conclusion of data analysis. The USB flash drive, stored in a locked container, contained all research data. After five years, the flash drive will be destroyed using the appropriate technology at that time.

Issues of Trustworthiness

Winter (2000) introduced the concept of a continuum of opinion on the issue of validity and the different definitions of the term validity within the qualitative research environment. Many researchers do not subscribe to validity with respect to qualitative research. Shenton (2004), in an effort to provide rigor and ensure the trustworthiness of a qualitative study, identified specific guidelines to support each of the four major criteria of trustworthiness: (a) *credibility* (relationship of results to reality), (b) *transferability* (a reader's ability to transfer information to their individual context), (c) *dependability* (attention to changing environment), and (d) *confirmability* (results confirmable by others). Subsequent subsections include discussions on each of these four components of trustworthiness.

Credibility

The criterion of credibility deals with the relationship of the response data and reality. Shenton (2004) put forth 14 guidelines to ensure that the key component of credibility will be met. These 14 guidelines, along with how the structure of the study supported 11 of the guidelines, are as follows: Adherence to current research methods,

familiarity with the culture (I have 25+ years in the industry), support for participants' candid responses (candidness addressed with participants), member-check (participant approval of transcript, concept map, and textual summary of concept map), prompts to support interview questions, checks and balances from superiors on both the project and analysis of response data, researcher reflection on participant response data (concept maps), researcher experience, random sampling (selection of pilot study participants), triangulation, findings from previous research, negative case analysis, and "thick description of phenomenon under scrutiny" (p. 69).

Two guidelines that I did not follow in this study were triangulation and negative case analysis. Triangulation entails the combination of multiple methods, for example interview, observation, or focus group. This study did not employ multiple methods, but rather relied solely on interviews. Negative case analysis implies that all concepts are considered. A one-off experience of a single participant, not echoed by other participants or related to the abstract concepts that emerged across all participants, was not included in the analysis.

A third guideline that I did not completely follow was random sampling. The strategy for selection of community partners consisted of contacting past clients for their participation in the study, a purposeful sample. However, once I identified the sample from the community partners, a random selection process identified the three pilot study participants.

Transferability

Because of small sample size, results from a qualitative study are not generalizable. Instead, qualitative studies strive for the criterion of transferability. Transferability is the ability for a reader to transfer the results data to the readers' own circumstances. The reader must be able to relate and connect the results data to the reader's environment and context (Patton, 2002). Transferability does not support strong generalizations about the results to a population, but rather provides the possibility for a reader to make connections in another discipline. This study is transferable to the government contracting community, DOD and non-DOD, as long as the boundaries defined in this research are maintained, specifically participant selection criteria and software-intensive system contracts. This study is not transferable to U.S. government contracts outside of those boundaries, for example the purchase of automotive vehicles.

Dependability

Contributing to the concept of trustworthiness, dependability places a responsibility on the researcher to thoroughly describe the research effort, including surrounding descriptions of the environment within which the study is conducted with attention to any changes that may have taken place. The environment is well defined for this study. Interviews were conducted by telephone with a participant secluded in a private room. All participants were currently managing or had previously managed large software-intensive development efforts. The focus of the research was the experiences of

the participants who had satisfied predefined selection criteria. Precise definitions of the boundaries of this study supported the dependability component of trustworthiness.

Confirmability

To ensure the confirmability component of trustworthiness, the member-check process proceeded as previously specified, prior to the migration of raw data to thematic constructs. The migration process consisted of repeatedly iterating through first cycle and second cycle coding methods. Migration is not a sequential process, but an iterative process through steps that are interrelated yet described sequentially. Additionally, my committee chair and committee member conducted periodic reviews of the data analysis process and products. Those reviews supported the requirement that others confirm research results.

Trustworthiness Summary

With respect to this study, the pilot study helped assure that the participants satisfied the criteria for participation. Additionally, the pilot study led to modifications to the wording of the semistructured interview questions to assure that the questions were relevant and elicited the desired response data.

When considering trustworthiness, this study is well documented, including but not limited to (a) Chapter 3 and the detailed specification of the steps performed in this study as well as the order in which the steps were conducted, (b) the questionnaire (see Appendix F) that defined the criteria for participation in the study, (c) the Interview Protocol (see Appendix G), and (d) the coding strategy. Lastly as designed, this study

supported the four components of trustworthiness, specifically credibility (believability from the participants lens), transferability (ability of a reader to relate the study to the reader's context), dependability (thorough description of research setting), and confirmability (raw data and results corroborated by others).

Ethical Considerations

The ethical issues encountered by researchers both before and during phenomenological research focus on the protection of the right to privacy for research participants (Walker, 2007). A researcher has multiple obligations to fulfill to protect the privacy of participants. The first obligation is compliance with the National Institutes of Health (NIH) Web-based training course "Protecting Human Research Participants." The IRB maintains a copy of my Certificate of Completion. A second obligation is to adhere to the guidance of the Walden University IRB.

With respect to adherence to IRB requirements, the IRB provided a detailed checklist that identified items a researcher must address to support an ethical approach to research. The IRB stipulated that a researcher specify the (a) protection of textual documents by storage in a locked container, (b) protection of electronic documents using password-protection, (c) templates for Letter of Cooperation and Consent Form, (d) a Research Ethics Planning Worksheet listing approximately 40 items to be addressed by a researcher, and (e) that the data is to be maintained for a period of 5 years. I have addressed those items within the body of this paper.

Additionally the IRB review examined the questionnaire and interview protocol used to ensure the instruments addressed practitioners in an objective and unbiased manner. As previously stated, these documents are provided as Appendices F and G.

In addition to the NIH PRHP and the Walden IRB requirements, there are two major guidelines, or principles, that ground the ethical considerations of any research study: beneficence and nonmaleficence (Walker, 2007). Beneficence is the assurance that researcher actions are carried out to help participants feel comfortable, safe, and protected. Nonmaleficence is the assurance that no harm shall come to the participant as a result of participation or researcher actions. To support the combination of these two concepts, I (a) left biases behind and conducted an objective analysis of the research data, (b) acted professionally and thoughtfully, and (c) protected all data relating to participant, organizational, and geographical identification either through password-protected files or storage in a locked container. In my research effort, there is neither harm nor risk to the participants.

Lastly, it remains the researcher's responsibility to carefully plan and execute all the steps required to protect the integrity of the research study and the participants' identity (Walker, 2007). Participant identification is confidential, not anonymous; researcher communication with participants relies heavily on email consequently knowledge of each participant's email address was required. I executed due diligence to inform the participants of the steps taken to ensure the protection of email correspondence and the protection of participant identification.

To ensure participant confidence in the researcher, not only must participants' email communications and identity be guaranteed, a bond based on trust must be established. With a phenomenological research study focused on the sharing of participant experiences with a researcher, the researcher is entering the participants' consciousness (Walker, 2007). A bond between the parties involved is necessary for the participant to fully realize the degree of personal privacy guaranteed by the researcher and for the researcher to gain candid response data throughout the sharing process. This bond was established by sharing with participants the (a) goal of the research, (b) purpose of the research, and (c) mechanisms for executing the research all of which contributed to supporting the ethics of conducting research (Walker, 2007).

Summary

The contents of this chapter document an approach, approved by the IRB (Approval Number 02-05-14-0269265), to my phenomenological research study to explore software program and project managers' experiences with assessing and reporting software development completion status. The problem is that existing monitoring mechanisms, including abstract truisms, traditional management techniques, and detailed syntactic metrics do not address completion status, leaving managers in a compromising position when reporting status to superiors and stakeholders. A phenomenological approach targeted the essence of the lived experiences of personnel who manage, monitor, and report completion status. The approach consisted of a set of well-defined steps, that when followed, led to the (a) acquisition of a sample, (b) a pilot

study and main research study, (d) raw response data, and (e) an approach for analyzing the raw data. Chapter 4 contains discussions with respect to the key findings that evolved from conducting the study according to the steps in this chapter.

Chapter 4: Results

The core research question, introduced in Chapter 1 and repeated here for convenience, was as follows: *What meaning do government contractors ascribe to their experiences with software metrics relevant to assessing and reporting software completion status?* Three research questions supported the core research question and are repeated in Table 17 for convenience. These questions focused on eliciting managers' positive and negative experiences of assessing and reporting software completion status with current measurement mechanisms. Results of analysis of participants' response data to the three supporting RQs are found in this chapter. Interpretation of the results with respect to the core research question and the three research questions is found in Chapter 5.

Table 17

Research Questions

Research questions
RQ1: How have current software metrics supported the assessment of software development completion status as perceived by program and project managers?
RQ2: How have current software metrics supported the reporting of software development completion status as perceived by program and project managers?
RQ3: What is the relevancy of software metrics to Software Development Life Cycle (SDLC) phases?

My research indicated that a significant amount of time, potentially not accounted for in a response to a request for proposals (RFP), is required to perform all the tasks

associated with assessing status. Consequently, reporting that status remains difficult. Managers' reliance on metrics varies widely and depends upon the phase of the SDLC: Reliance on metrics increases as a project progresses through a SDLC; however, insight into true completion status remains elusive. The following subsections contain a summary of the pilot study, the demographic analysis, a summary of the data capture and data analysis methodologies, trustworthiness, the results of the analysis, and a summary of the research study.

Pilot Study

Upon identification of the participants and the generation of the participant mapping table (see Table 14), a random selection process identified three participants to participate in the pilot study. The process consisted of selecting randomly three pieces of paper, without replacement, from a hat containing 20 pieces of paper, each identified with a unique participant ID number. The pilot study served four purposes: (a) to validate that the participants satisfied the selection criteria, (b) to analyze pilot study participants' raw data, (c) to ensure that the interview questions elicited relevant response data, and (d) to confirm the member-check process.

In summary, the pilot study provided the desired results. The demographic analysis validated participant selection criteria. With respect to experiential raw data, the interview questions elicited data relevant for analysis. Lastly, the combination of the concept map and transcript summary, along with the sanitized interview transcript, supported a thorough member-check process.

Pilot Study Process

The pilot study followed the steps specified in Appendix H and the IRB application. In summary, a third-party vendor, NoNotes.com, recorded and transcribed interview content. I converted the raw unsanitized data provided by NoNotes.com to sanitized MS Word files. A sample transcript representing the sanitized transcription of one interview, specifically the interview for Participant P10, is provided in Appendix B. Using Microsoft™ 2011 PowerPoint, I created a concept map for each participant's transcript to (a) facilitate subsequent data analysis, (b) evaluate the relevance of interview protocol questions to the research, and (c) assess the credibility of the raw response data. These concept maps then formed the basis for a textual summary contained in the Notes section of a PowerPoint slide.

A sample concept map, representing the concept map for participant P10's transcript (see Appendix B), is included as Appendix I. The corresponding textual summary for P10's concept map is provided as Appendix J. I provided three documents—sanitized transcript, concept map, and summary—to each of the three participants in the pilot study for validation of content. The member-check process continued until each of the pilot study participants provided an email indicating agreement with my interpretation of the interview content.

Analysis of the pilot study work artifacts provided an initial set of nodes in NVivo v10. Iteration within and between codes and nodes using the open and axial coding process continued until the resulting codes and nodes converged to a stable set that

supported a synthesis of each of the pilot study participants' response data. These pilot study codes and nodes formed the baseline, or initial framework, for the main research study. Refinement of these nodes continued throughout the analysis of all raw data.

Impact on Main Study

Analysis of the pilot study results impacted the main research study instruments in the following areas: questionnaire, interview protocol, and participant identification schema. Minor changes to the questionnaire and the interview protocol improved the quality of the research going forward into the main study. I modified the participant numbering schema to a sequential numbering schema (e.g., P1, P2, . . . P20) to accommodate the fact that participant numbering must take place before a sample can be drawn.

Setting

The settings for both the pilot and main research studies were identical. The transcripts support the fact that the participants were uninterrupted during the interview. I forwarded a copy of the interview protocol to all participants at least 48 hours in advance with a request to look it over and annotate their document with concepts that they would like to include in their responses. All but two participants took the time to organize their thoughts, resulting in focused interviews with little need to return a participant's focus back to a specific question.

I placed a call from my office using the NoNotes.com application on my iPhone 4 or landline (see Appendix K). In summary, the setting for both researcher and participants was straightforward and uncomplicated.

Demographics

Four community partners who contract to U.S. government agencies (DOD and non-DOD) agreed to participate in this research and provided a total of 20 participants. I used SPSS v21 to conduct demographic analysis on the data collected by the questionnaires. The codebook associated with this study is included as Appendix L.

Prior to running the descriptive statistics on the demographics, I visually inspected the data for invalid or missing data. All 20 participants responded to all demographic questions and provided valid data (see Table 18). Subsequent subsections provide specific results for frequency distributions as well as central tendency statistics for years of experience.

Table 18

Validation of 100% coverage

		yrs_ Exp	gvt_ Agncy	type_ Gvt_Agncy	Role	CMMI_ Lvl	rpt_ Int	rpt_ Ext	Gender
N	Valid	20	20	20	20	20	20	20	20
	Missing	0	0	0	0	0	0	0	0

Frequency distributions. Sixty percent of the participants were project managers, and the remaining 40% were program managers. Eighty-five percent of the sample, or 17 participants, were affiliated with CMMI Level 3, while the remaining 15% preferred not to provide that information. With respect to the gender of the participants,

men comprised 85% of the sample, while three women comprised 15%. Two of the women were project managers, and one woman was a program manager. Twelve participants were involved with DOD applications, with the remaining eight participants affiliated with non-DOD government agency applications. All participants reported software development status internally within their organization, and all but two participants had experience reporting software development status externally to stakeholders. These two participants had major roles within their organization, having responsibility for a software factory to support both internal organizational and customer needs.

Descriptive statistics: Years of experience. The mean years of experience was 16.50 years (*SD* = 9.058). The values 8.50, 15.0, and 23.75 years of experience represent the 25th, 50th, and 75th quartiles respectively. The mode was 15 years of experience. The values ranged from 2 years of experience to 33 years of experience resulting in a range of 31 years.

Data Collection

The process of raw data capture, transcription, and analysis was previously specified in Chapter 3 and Appendices F, G, and H. Responses to questionnaires provided demographic data. Transcripts of interviews consisted of semistructured interview questions. Interviews, conducted by telephone and recorded/transcribed using the third party contractor NoNotes.com, provided experiential data. The data collection process followed the steps discussed in Chapter 3 and included in Appendix H.

Detailed audit trails, implemented as Microsoft™ Excel worksheets, captured the distribution/agreement dates for research artifacts (e.g., questionnaires, interviews, concept maps, summaries) as part of the member-check process. One sample audit trail, specifically for capturing interview date, start/stop time, and duration, is included as Appendix M.

Demographic data collection began on February 8, 2014 and ended on March 11, 2014. Interview data collection for the pilot study began 14 March 2014 and ended on 19 March 2014. Interview data collection for the main research study began on 1 April 2014 and member-check ended on 23 April 2014. The average interview duration was 40 minutes. When necessary I contacted participants by email to clarify transcript content. I sanitized the unsanitized transcript data before submission to participants for approval. Upon approval of concept maps and summaries targeting the transcript data, I migrated the transcripts to NVivo v10 for analysis.

Data Analysis

The pilot study concept maps and transcriptions led to an initial set of nodes in NVivo. The initial node count after the pilot study was seven Level 1 nodes, seven Level 2 nodes, and eleven Level 3 nodes. The structure of the node hierarchy followed the structure of the research questions: assessment (positive, challenges, pain point), reporting (positive, challenges, pain point), and SDLC. Shortly, it became clear that words—for example time, engagement, and understanding—crossed node boundaries. While analyzing this issue and deciding what approach to take, I isolated all participants'

response data to the questions relating to RQ3, the relevancy of metrics to SDLC and SDLC phases. I analyzed the response data to those five questions, specifically Questions 8–12 inclusive (see Appendix G), using a tabular format. These questions resulted in more targeted and focused responses than the response data for RQ1 and RQ2. This focus enabled analysis without the use of NVivo. I read the responses, took copious notes, formatted the notes into a tabular format. The tabular format allowed for the calculations of percentages that contributed to the key findings for RQ3. I then created a concept map to represent the model for the raw response data to the five questions supporting RQ3. For the remainder of the response data, specifically the response data related to RQ1 and RQ2, I used NVivio.

Because the node structure of NVivo was based on RQs and interview questions, concepts like time, understanding, engagement, and differences can coexist in subnodes for RQ1 and RQ2 with identical names. For example, although time occurs as a subnode in each of the higher-level nodes for RQ1 and RQ2, the data coded to those subnodes comes from responses to RQ1 and RQ2 respectively.

Evidence of Trustworthiness

As stated in Chapter 3, there are four major components of trustworthiness: transferability (reader's ability to relate study to self), credibility (participant approval), dependability (reader must comprehend the research effort), and confirmability (confirmation of results). Shenton (2004) drew parallels to the four components of quantitative, or naturalistic, research as illustrated in Table 19. Each of the four

components of qualitative research specified in Table 19 is critical for a researcher producing, and the reader consuming, a qualitative study. Stated differently, the producer/consumer paradigm is appropriate when considering the

Table 19

Comparison of Quantitative and Qualitative Validity Components

Quantitative and qualitative components compared			
Quantitative	Meaning	Qualitative/Naturalistic	Meaning
Internal validity	Relevance of what is being measured to the measurand	Credibility	The reality of the phenomenon being researched
External validity	Generalization	Transferability	Context must be well defined to draw parallels
Reliability	Repeatability with similar results	Dependability	Replication of research process. Changing phenomenon may yield different results
Objectivity	Reduction of researcher bias	Confirmability	Ensure bias is identified and managed by researcher & member-checks

Note. Data abstracted from Shenton, A.K. (2004). *Strategies for ensuring trustworthiness in qualitative research*

validity, or trustworthiness, of a qualitative study. Subsequent paragraphs contain discussions with respect to how my research satisfied each of these four components.

Transferability

Due to generally small sample sizes, qualitative results are not generalizable. For qualitative research, transferability is the closest equivalent to generalizability (Shelton, 2004). The researcher must provide necessary and sufficient information for a reader to make connections between the research study and the reader's context and experiences.

Using recommendations from Shenton (2004), the following (a) support the elements of transferability for a reader and (b) indicate that this study is transferable.

- Four organizations, geographically distributed across the United States, participated in the study and all were contracted to government agencies.
- Participants had at least two years of experience managing DOD or non-DOD software intensive efforts.
- There were 20 participants with the demographics identified in a previous section titled Demographics.
- Questionnaires and interviews supported the collection of demographic and experiential data.

Credibility

There is no mathematical or scientific formula that is applicable to qualitative data analysis. To support the credibility of the study, the research method followed an iterative process of open coding followed by axial coding facilitated by Saldaña's (2012) first and second cycle coding methods. The decision whether to identify nodes prior to research (a priori) or emerge during analysis (emergent) is researcher dependent (QSR International, 2013). I specifically did not identify any codes prior to data collection, but rather let the codes emerge from transcripts and concept maps. This decision was made in an effort to reduce bias. I have more than two decades of experience with the phenomenon being researched and needed to focus solely on the transcripts/concept maps to reduce bias.

With respect to researcher bias, maintaining a semiscientific perspective to the coding process also mitigated bias. Specifically, I focused on the coding methods as specified by Saldaña (2013) that supported an emphasis on a more structured, rather than a completely subjective, approach to identifying codes and patterns, all conducted within an open and axial coding umbrella because coding is never sequential but rather cyclical in nature (see Figure 8).

Dependability

The changing nature of a phenomenon makes it difficult to repeat a phenomenological research effort and achieve similar results; however, credibility is directly proportional to dependability (Lincoln & Guba, 1985). A higher level of credibility results in a higher level of dependability. Consequently a full and detailed presentation of the research process used is essential to support dependability of the analysis of a static phenomenon. Chapter 3 and Appendix H contain descriptions of the step-by-step process that was followed for this study.

Confirmability

I made a conscious effort to ensure that the interpretation of the raw data was a true representation of participants' content. Each participant reviewed, and commented on, (a) the transcript, (b) the content of the transcript represented diagrammatically as a concept map, and (c) a textual summary written to the concept map. The process of synthesizing these final products led to a progression of the concepts from each individual's interview up to the final analysis models.

Results

Data analysis resulted in three major findings, and their associated subthemes. Each major finding corresponds to a research question. Each of the three research questions and the corresponding analysis results are discussed in the following subsections. Within the following discussion, a participant's ID, assigned in accordance with the schema defined in Table 14, identifies the source of a participant's quote. Participant numbers initially ranged from 1 to 24, however the following ID numbers—specifically P2, P3, P16, and P18—do not appear in the analysis of response data. Those participant IDs represent the four participants who withdrew from the study.

RQ1: Assessing Software Status with Current Measurement Mechanisms

A synthesis of participants' responses showed that activities required in a metrics effort, even if informal, are time-intensive. The concept map for this theme is depicted in Figure 10, where the circle highlights the number of relationships to the concept time.

Subtheme 1: The time-intensive level of effort associated with metrics.

Metric collection "does take an amount of effort and a lot of people, particularly engineers who don't like that part" (P12). Some engineers do not like to be held accountable and P15 stated, "they'll just keep feeding you data to make you go away." With respect to the frequency of metric collection, formal monthly reviews and weekly informal reviews are manageable. However, reporting "on a daily basis" is going to be difficult (P12). P23 added that it was "overkill" and expressed concern over how to "get some of the work done." Time is needed to synthesize metrics from different tools,

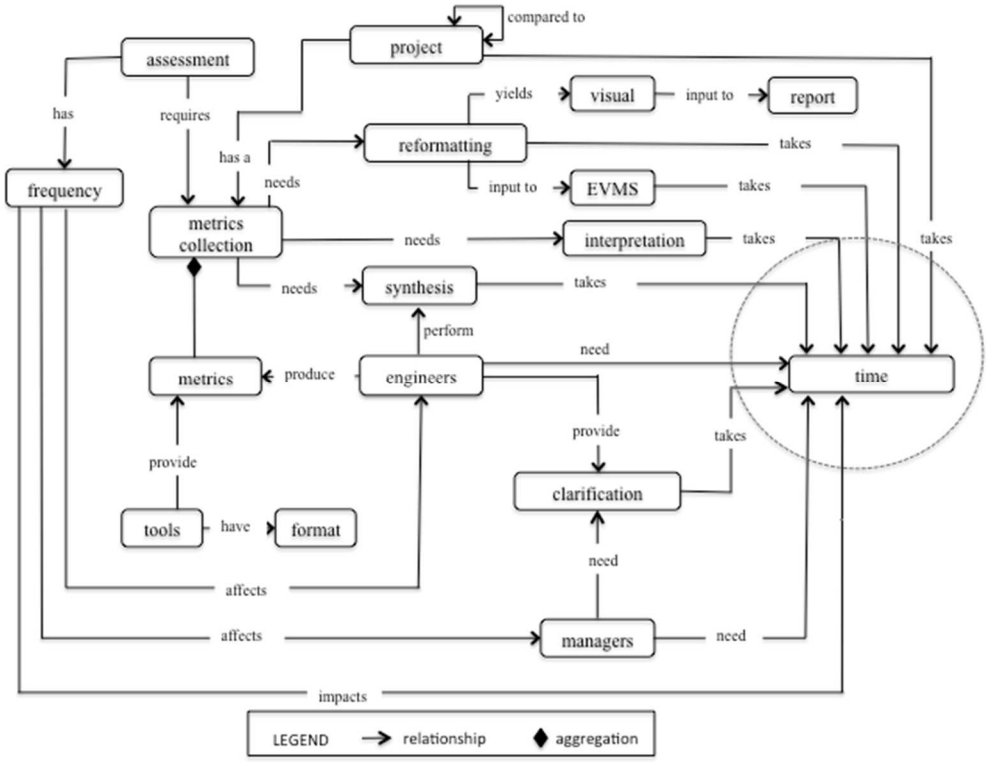


Figure 10. Concept map: Time-intensive nature of assessing software status (figure created by P. Texel using Microsoft PowerPoint 2011).

specifically on "integration efforts where you take a lot of commercial off the shelf, government off the shelf, all of these third party tools and put systems together" (P23). "We need better tools, I do not have what I need to assign an efficiency weight to engineers to improve estimates" (P23). Time is required to develop improved metric collection tools. All these activities take time. Add to that the time it takes to synthesize and reformat metrics from the format and content provided by multiple tools to a comprehensive simple visual representation of the numbers that is easy to look at "but doesn't tell the whole story" (P23). Considering all the issues surrounding metrics, P7 asked, "Do I really know where I am? ", and then stated, "I always had the feeling that I

was unsure that I really knew where we were" (P7). This lens was echoed by P13 who asked, "What do these metrics really mean?"

Subtheme 2: Interpreting metrics is time-intensive. Participants stressed the time required to understand and interpret what the numbers actually mean. Seventy-five percent (75%) of the participants indicated that understanding was an issue. Stated differently a "holistic approach" (P5) to metrics is required. When metrics do not make sense "you go digging into why this does not makes sense" (P21), you "go to the engineer to dig in further" (P17) and that takes time. Once understood, it takes more time to roll up the metric data to report to internal management and stakeholders. P9 stated, "I don't want to use the word whitewash" but those numbers "better look good when simplified." Often, you need to "tailor the message" (P5),"be crafty" (P5), and "make sure the numbers do not turn red" (P6). "You need to have your story before you go and report it internally if you want to keep your job" (P21).

Interpretations can lead to different values for metrics. Stated differently, when engineers use different definitions to generate a metric, the result is different values for the metric. For example, consider the definition of a defect. One engineer might call something a defect and the next one might not and so that was where metrics became skewed" (P22). The definition of what is to be counted to achieve a number/metric is "ambiguous" (P21) and results in different values of the metric. Resolving which definition to use takes time.

Subtheme 3: The incompatibility of metrics with Earned Value Management System (EVMS). Thirty percent, or six participants, had experience with agile software development processes. Of those six, four were involved with non-DOD projects, while the remaining two were involved with DOD projects. Four were project managers and two were program managers. There was no difference in the experiences of all six participants. All six participants expressed difficulty with merging the data with de facto management tools, specifically Microsoft Project and/or EVMS.

Significant time is required to provide a mental bridge between metrics and the input requirements of EVMS when using an agile process. Program managers expressed frustration when "using EVMS" and they "check the box to get a schedule out" (P5). P19 stated that there were "considerable challenges in that regard." The incompatibility of agile metrics with EVMS creates "a painstaking process that my team spent several hours trying to figure out" (P17). Time is needed to "understand how to kind of mold the two paradigms together" (P17)—agile and EVMS.

Additionally, when using an OO approach to development there is an issue when monitoring progress and performance. The difficulty is relating the percentage of classes complete to "an integrated master schedule" (P19). On a first project, it was "an enormous challenge" to correlate the OO metrics back to the original cost and schedule estimate (P19).

Subtheme 4: The inability and time required to compare current and past metrics from multiple projects. With respect to tools, drawing comparisons across

projects takes time. When comparing metrics to metrics from past programs, for example to compare efficiency or to support estimation tasks, time is involved. The problem is that not all programs report metrics according to the same criteria. P14 stated, "You have a subjective tool based on perhaps past experience, which could be shaky. It's not guaranteed that even within one organization, that their past history is meaningful." For example, a "count could include configuration files while on another project they don't count configuration files" (P4). A significant amount of time is required to obtain a fair comparison; time to research the metrics collection inclusion criteria and time to identify the specific counting rules. "Frankly it is a mess" (P14).

This same issue arose when comparing programs in areas other than programming metrics. "The finance people, they always like you to use a similar tool method on your basis of estimates which means that you need to find another program that is similar and find out how many hours of development that they used" (P9). Again, comparing numbers from other programs is problematic.

Subtheme 5: Indirect responses. P5 introduced one additional aspect, specifically estimates. P5 indicated, "I rarely see actual projects aligned to that (estimates based on metrics) very well." P14 stated the feeling of assessing software was "like being in a maze." Software development traverses a particular path. A manager experiences comfort with a path, only to later learn that the selected path is not a valid path through the maze. P8 highlighted the difference between current status and where the procurer wanted the status to be. Direct, as opposed to indirect, responses to the

research question were not forthcoming. One exception was a statement made by participant P13. P13 wanted to know what the metrics really mean.

RQ1: Summary. Synthesis of the response data indicated that current metrics do not support the assessment of software completion status. This conclusion is supported by the analysis of the response data for RQ1. Metrics are ambiguous, subject to interpretation, and must be transformed into a linear management tool (e.g., EVMS, Microsoft Project). Participants specified the need to create a story based on the metrics and stated that the story better be a good story. One common topic that did occur among all participants' responses was the concept of time.

There is an inordinate amount of time required to collect metrics from engineers, synthesize metrics from different tools, interpret metrics to obtain a holistic view, rollup metrics for management and customers, reformat metrics for presentation, and analyze metrics from past programs for comparison or estimation purposes. Numbers on a piece of paper are just that, numbers. Without context, misinterpretation is both possible and probable. Compounded by the frequency of status assessment, the key issue identified in my research was whether sufficient time to execute a metrics process had been included in a proposal.

RQ2: Reporting Software Status with Current Measurement Mechanisms

Analysis of the response data for RQ2 indicated that reporting software status is a time-intensive activity that may not be fully realized. Figure 11 represents the concept map for this theme. The assumption made in Figure 11, and in the following discussion,

is that the metrics for assessing status have been collected, synthesized, and are ready to report. Experiences reporting those metrics are the focus of this research question.

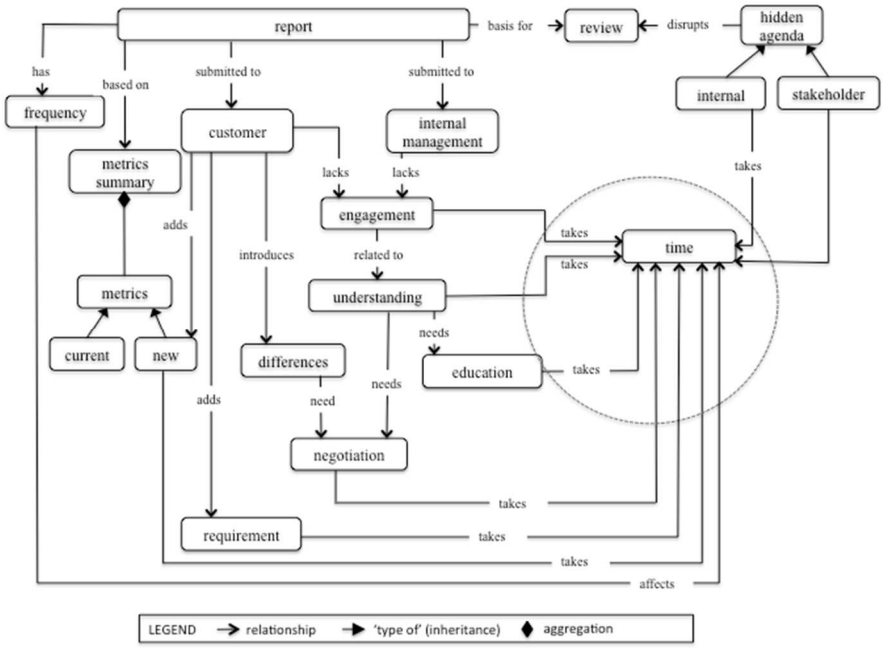


Figure 11. Concept map: Time-intensive nature reporting metrics (figure created by P. Texel using Microsoft PowerPoint 2011).

Subtheme 1: The time-intensive level of effort associated with reporting metrics. Reporting software status is just as time-intensive as assessing software status. The *frequency* of reports *impacts* time. An increase in the number of reporting cycles results in a corresponding increase in time to support the reporting process: time to participate in meetings, subsequent discussions, and telephone calls for clarification. The success of these meetings, discussions, and follow-up telephone calls are hampered by lack of customer engagement. "Sometimes my customer wasn't engaged as much as he should have been " and "had the old waterfall mentality" (P17). Often but not always, there is a corresponding lack of understanding of software development. Engagement

was a significant issue for seven of the participants, or 35%. Five of the seven participants were project managers and two of the seven were program managers.

> The challenge is trying to get them to understand that no, you know what you are talking about, and you are not trying to sit on a charge number, and you are not bluffing them, and you are not putting them off. It's God's honest truth what you are reporting. (P12)

Subtheme 2: Multiple differences between contractor and stakeholders require time to negotiate an agreement. When reporting software status to stakeholders many differences become apparent. One difference is the difference between the metrics "expected versus what the project is reporting" (P1). P10 stated, "That is a challenge in terms of the cost and schedule and duration and time, it stresses these." For example, when a customer is expecting specific metrics (e.g., SLOC) and SLOC is not presented, customer expectations are not met. When the expectations of the customer regarding metrics are not met because "they understand metrics in one form" and they are being reported differently, time is needed to educate to achieve a level of understanding (P1).

Differences also exist between what is reported and what is reality, for example "whitewashing" (P9). P10 stated that there is "always an undercurrent of things that are not being discussed." Looking at the criteria for successfully passing a review, "you have to grade fairly" and that grading is "subjective" thus creating differences (P9). Various components of a review (e.g., architecture components) need to be graded at the same

level. If these components are not graded at the same level, "passing a review may be too easy" (P9). A project can transition to the next phase with the potential for problems in later phases due to work that was not completed in a prior review. Stated differently, an "overemphasis on reporting work done in the early stages" results in the fact that "you have to catch up later" (P9). This domino effect can impact both cost and schedule. Additionally, consider whether or not a specific milestone has been reached. If "exit criteria have been watered down, weakened" (P1), then there is a difference between reported status and actual status. Because of the "domino effect" these types of gaps need to be closed somewhere in the project or future completion is jeopardized (P1).

There are differences in the level of stakeholders' level of understanding. "I still don't think our division understands metrics, our customers don't really understand software" (P4). This difference can result in the software metrics portion of a review being "just skipped over" (P4), or if presented, "you start to lose them" (P21) and "their eyes kind of roll at the back of their heads" (P4).

An additional challenge is the time it takes to get the customer to understand what the metrics mean. Explaining the difference between equivalent lines of code (ELOC) and delivered lines of code (DLOC) proved "painful" for one participant (P21). Time is also needed to educate management and customers to understand that it is not possible to give an accurate completion date for a software interface to missing or delayed hardware. The impact of this inability to provide a definitive date cannot be underestimated.

Management and stakeholders' responses to the lack of concrete data are "not good enough" (P5).

Subtheme 3: Hidden agendas are a silent consumer of time. Meetings are disrupted when meeting participants have hidden agendas. Political agendas can exist within a corporation. Previously it has been stated that numbers alone are not sufficient to analyze metrics, that metrics are best understood with a holistic view, and that there is a difference between the numbers themselves and what the numbers actually mean.

When audits of projects are conducted for the purpose of, for example an organizational meta-analysis, numbers at a meta-level lack the accompanying holistic view of those numbers. P5 stated that the result can be very "demoralizing " for the project/program manager for that effort and can be "taken against you personally or against your career."

RQ2: Summary. Current metrics do not support the reporting of software completion status. P13 wanted to know what the metrics "really mean" because different people can interpret them differently. As with assessment, participants expressed the need to craft a story, ensure the numbers look good, and expressed concern over what was not reported.

As with assessment, there is an inordinate amount of time involved in reporting metrics to management and stakeholders, from the frequency of reports (e.g., daily, weekly, monthly) to negotiating agreements on differences based on expectations and reality. This is exacerbated by the fact that often metrics are presented to management

and stakeholders as numbers, in a table, spreadsheet, or rolled up in a dashboard. An explanation of where the numbers come from and what the numbers mean is missing, thus presenters become easy targets for those with hidden agendas. These differences, due to lack of engagement and/or lack of understanding, can be mitigated with increased engagement and education, both of which take time.

RQ3: Relevancy of Software Metrics to SDLC Phases

Figure 12 represents the concept map synthesizing participants' responses to this research question. Two subthemes, subsequently addressed, emerged from this concept map and are delineated by the horizontal dashed line in Figure 12.

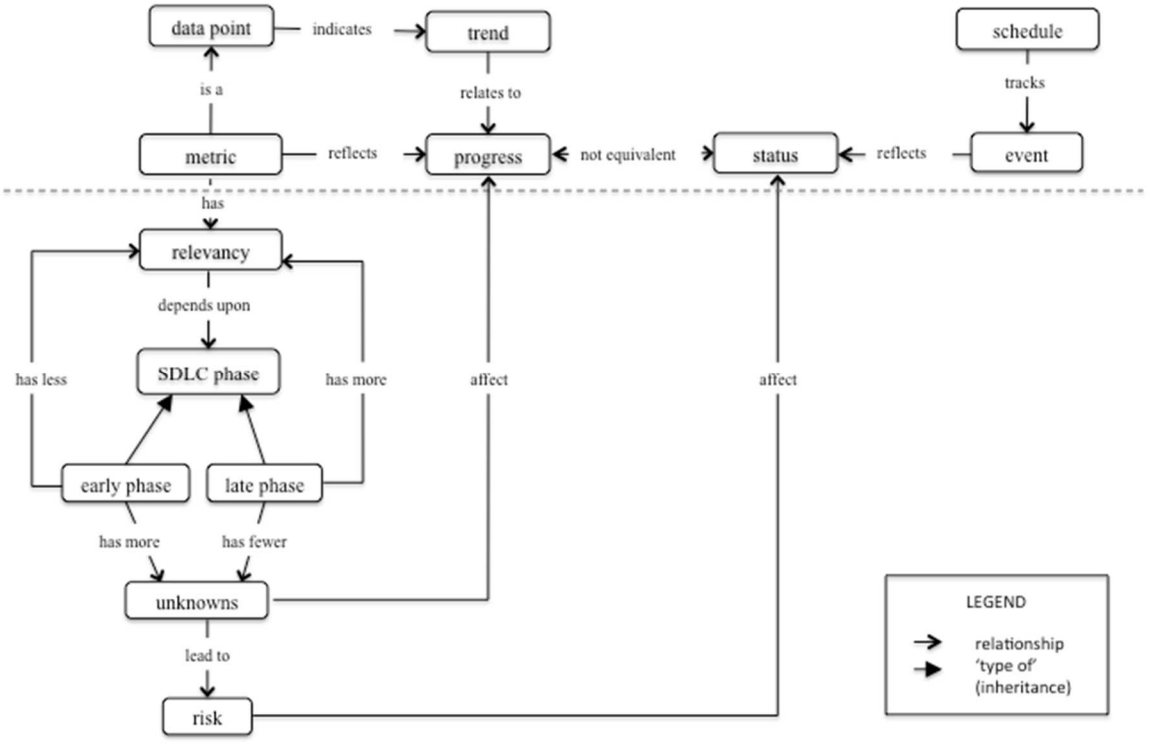


Figure 12. Concept map: Unknowns and risk affect the relevancy of metrics (figure created by P. Texel using Microsoft PowerPoint 2011).

Subtheme 1: Progress and status are not equivalent. The first subtheme to emerge from this concept map, illustrated in the top half of Figure 12, relates to the use of the words *progress* and *status*. Participants used these two words interchangeably and the two words are not equivalent. Progress is forward movement towards some pre-defined goal, while status is the current position, or state, at some moment in time. Status and progress are two different words, with two different meanings. Increasing SLOC provides an indication of progress. Events on a Gantt chart reflect event start and stop dates, initiation and completion, but not progress within those dates. Metrics represent "data points, not answers" (P15), data points indicate trends, and trends relate to progress, but not necessarily to completion status.

Subtheme 2: Relevancy of metrics to SDLC phase. The second subtheme, graphically portrayed in the lower half of Figure 12, relates to the fact that a SDLC can be viewed in terms of phases, specifically phases before and after Critical Design Review (CDR). Phases of the life cycle prior to CDR, pre-CDR, are characterized as "there are just too many unknowns early on" and later phases of the life cycle (post-CDR), are characterized by the fact that "the risks seem to reduce" (P10). Unknowns result in delayed progress while risks can result in project failure.

According to the majority of participants, metrics are less relevant in the early phases of the SDLC than those in the later phases. Although predominant, not all participants shared this view. What follows is a discussion of the percentages that contribute to this subtheme, as well as those experiences that differ from the majority.

SDLC phases with the least relevant metrics. When asked what phase of a SDLC had the least relevant metrics, there was a general trend in that 70% of the participants responded that metrics in the early phases of the life cycle (pre-CDR), were of little to no use, specifically they were "open-ended" (P5), "a lot of on the fence words" (P5), "ambiguous and open to interpretation" (P12), or "cloudy" (P7). Of that 70%, 45% specifically identified the requirements phase, an interesting finding explored in Chapter 5.

As previously stated 70% of the participants specified pre-CDR phases had the least relevant metrics. The remaining 30% of participants who did not select a pre-CDR phase as the least relevant specified (a) testing (15%) due to lack of confidence in the quality of individual test cases, "was the test valid" (P13), and test case coverage, (b) coding (10%) indicating that code is "constantly in flux" (P4), and (c) a view of no distinction between phases (5%).

SDLC phases with the most relevant metrics. With respect to the most relevant phases of the life cycle, 85% of the participants' indicated that the metrics provided in post-CDR phases were the most relevant, with 55% indicating testing, 25% selecting the coding phase, and 5% simply indicated post-CDR, not selecting a specific phase. Of the remaining 15%, 5% of the participants' specified design and 10% viewed all metrics at the same level, indicating that satisfying SDLC phase entry and exit criteria were more critical than metrics (P1).

RQ3: Summary. On the issue of the relevancy of metrics across *all* phases of the SDLC, the majority of participants (70%) expressed the opinion that pre-CDR phase metrics were the least relevant while 85% of the participants indicated that post-CDR phase metrics were the most relevant. Experiences with respect to *specific* phases of the SDLC exhibited a wide variance, with neither agreement nor majority opinion.

Revisiting the relevancy of metrics across all SDLC phases, the pre-CDR versus post-CDR majority view, there is one main contributing factor to this result. The contributing factor is unknowns. Unknowns, affecting progress, are also related to risk, and, as shown in Figure 12, unknowns can indirectly affect status. As a project progresses through a SDLC the number of unknowns decrease which leads to more confidence in the metrics in the later phases of a SDLC. The conclusion is therefore that there is less confidence in the metrics in the early phases of a SDLC where the number of unknowns is greatest, while there is more confidence in the later phases of a SDLC due to a decrease in the number of unknowns.

Results Summary

I conducted a phenomenological study according to the process described in Chapter 3 and Appendix H. This research study focused on the experiences of program and project managers associated with software intensive efforts under contract to DOD and non-DOD government agencies. Four contractors and 20 participants supported the study.

I analyzed the interview transcripts using open and axial coding, supported by concept maps and textual summaries. Analysis resulted in the fact that current metrics are not providing increased insight into software development, previously cited as a necessary step to contain cost and schedule overruns (NDIA, 2010; Whitfield, 2007; The White House, Office of the Press Secretary, 2009). Additionally a significant amount of time, thus cost, is devoted to assessing and reporting software status. However, as supported by the literature review and participants' experiences, the metrics lack relevancy to software status. Consequently, costly resources are being applied to metrics that are not providing the required level of management insight.

There is some promise with agile processes. However, Tarhan and Yilmaz (2014) cautioned managers to emphasize objectivity over subjectivity. An additional concern with an emergent technology, like agile, is for teams and individuals to create new metrics, thus replicating the proliferation of metrics identified for waterfall and OO processes identified in Appendix A (Misra & Omorodion, 2011).

The participants in this study voiced that the de facto way of conducting business is "broken" (P14, P15, P23), "rigid" (P9, P10, P14, P7), and "expensive" (P9), but is followed because that is the way the current process works. The lack of agreement on which phases of the SDLC are the most and least relevant to status is very telling. The message is that the choice is dependent upon personal experience and personal preference perhaps based on comfort with a specific phase, not on any objective approach that could provide consistency for all stakeholders for all phases.

This chapter contains material targeting the analysis of response data organized by the three supporting research questions, RQs 1, 2, and 3. Chapter 5 contains a discussion of the interpretation of these results with respect to the core research question. Additionally Chapter 5 provides a discussion of areas for future research.

Chapter 5: Discussion, Conclusions, and Recommendations

My research focused on exploring managers' experiences, while under contract to the U.S. government, of assessing and reporting software status on software-intensive systems. The intent of the study was to address the gap that exists between seminal authors'/researchers' approaches to software status measurement and the practical needs of managers who assess and report software status internally and externally. As previously stated, the 2010 NDIA report cited the need for management to improve its effectiveness through increased insight into software development (NDIA, 2010). The report also highlighted the fact that measures and indicators were not available to support that mission. My study indicated that not much has changed since the NDIA report.

Key findings of my study, related to research questions, follow:

- Key finding (RQ1): Assessment of software completion status is replete with difficulties. The difficulties include the time required to (a) collect metrics, (b) synthesize tool-specific formats, (c) interpret metrics, (d) rollup metrics for presentation, (e) migrate metrics to linear-based management tools, and (f) educate those who have not been engaged and/or do not understand software development. Unfortunately, these tasks do not contribute to managerial insight into software development completion status and lead to additional difficulties when using data from multiple projects for estimation and comparison purposes.

- Key finding (RQ2): There are multiple difficulties associated with reporting software status. The difficulties include the time required to (a) build a story describing the metric data, (b) educate those who have not been engaged and/or do not understand software development, (c) manage differences between expectations and reality, (d) address subjective versus objective measures, (e) handle concerns over what is not being reported, and (f) realize that progress is not equivalent to status. These difficulties make reporting software status unpredictable, time-consuming, and potentially questionable.
- Key finding (RQ3): Although not unanimous, participants generally viewed the metrics in the pre-CDR phases of a life cycle as the least relevant to status and metrics in the post-CDR phases of a life cycle the most relevant. However, with respect to specific life cycle phases, there was wide variation with respect to which specific phase of the life cycle had the least and most relevant metrics.

Interpretation of the Findings

Recall the core research question: *What meaning do government contractors ascribe to their experiences with software metrics relevant to assessing and reporting software completion status?* When taken together, the key findings indicate that the current metrics and measurement mechanisms do not support the assessment and subsequent reporting of software completion status. The assessment and reporting of

software status is time intensive and fraught with multiple difficulties. The end result is a data set that is irrelevant to monitoring completion status.

The simultaneous usage of the words *progress* and *status* by participants is an example of a communication disconnect. If management and stakeholders expect completion status and the contractor provides progress and/or technical characteristics, then communication fails due to lack of clarification of terms. Additionally, these data are migrated into software management applications (e.g., Microsoft Project, EVMS) that are based on a linear approach to development. Progress is not linear with software development efforts, but rather consists of iterations of progress, plateaus, and setbacks. Lastly, the time it takes to complete this repeating cycle of collection and reporting of data with little relevance to status is expensive, does not address status, and leaves less time to focus on the demands of the project.

With respect to the relevancy of metrics to an SDLC, the findings that pre-CDR and post-CDR phases were viewed as the least and most relevant phases, respectively, were not surprising. Unknowns tend to decrease as a project progresses through a SDLC. Concerning which *specific* phase of a SDLC had the least and most relevant metrics, the results did not support a majority view. The findings represented individual experiences with respect to a single phase, and those experiences are subjective.

One result of the analysis is striking. Forty-five percent of the participants identified the requirements phase as the phase with the least relevant metrics. The requirements phase is considered one of the most important, if not the most important,

phase of a SDLC. Requirements directly affect each phase of an SDLC and must reflect the business goal (Hull et al., 2011). An analysis of causal factors for overruns, published by the Standish Group (1995), indicated that poorly identified requirements were responsible for 48.1% of the overruns. Additionally, requirements creep is repeatedly identified as one of three most common contributing factors leading to overruns (Dev & Awathi, 2012; Hull et al., 2011). With 45% of the participants indicating the ineffectiveness of the requirements metrics, this is troubling. On the one hand, the requirements phase has been identified as one of the three most frequently cited contributing factors to overruns, yet the participants indicated that the metrics for the requirement phase are the least relevant.

The literature and the results of my study support the identification of the following differences that currently exist in the software measurement community:

- Definitions of basic measurement terms: *measure*, *metric*, *indicator*
- Operationalization of current metrics
- Validation criteria
- Values generated by tools on identical source code
- Programmer productivity related to application type
- Interpretation of metrics
- Relevance of metrics to specific SDLC phases

Additionally, different inclusion rules for counting, different definitions of what is to be counted, different tool sets, and different application types make comparison and

estimation based on historical data difficult, if not meaningless. The questions to be asked are (a) How are realistic and accurate estimates created and compared within an organization? and (b) How does the government compare responses to a Request For Proposal (RFP) from different contractors when variation in numbers between contractors is a given and the numbers are based on organizational metadata that are questionable in their own right?

These questions are applicable whether development is waterfall, iterative, incremental, or agile. Due to unexpected participant comments with respect to agile processes, I conducted additional research on agile metrics during the analysis of response data and found that there was no agreement on agile metrics. Data relevant to agile metrics had a wide variation. Metrics ranged from detailed syntactic metrics, such as SLOC, to management metrics, such as risk. Lastly, *definition of done* is a concept in agile processes and is defined as a measure of when a subset of functionality (a *sprint*) has been completed (Laanti, 2008). However, as stated by Davis (2013), there is "no operational guidance" on how to implement the definition (p. 165). Consequently, operationalizations of the definition of done can vary from project to project. More importantly, definition of done can be applied to more than one component of an agile process, such as sprint, *user story* (functionality stated from the user's perspective), or release (Davis, 2013). Again, consistency is lacking. Interpretation and implementation are organizational and project specific.

Software engineering is perhaps the engineering community with the least amount of rigor with respect to measurement and validation (Abran, 2010; Meneely et al., 2012). As previously stated, the literature review highlighted lack of agreement and inconsistencies in many areas of software development. My research indicated identical practitioner experiences with current management mechanisms used to assess and report status. What is clear is that although in some cases majority perspectives have been identified, there are multiple instances where there are inconsistencies and differences.

Limitations of the Study

One limitation to this study was the level of detail with respect to the identification of agile metrics when compared to the level of detail provided for waterfall, incremental, and iterative process metrics (see Appendix A). As previously stated, an unexpected outcome of the study was the introduction of agile processes in the responses of some participants. Forty percent of the participants addressed agile processes: some participants had actual experience and others did not, citing personal opinions. I conducted additional research and those papers have been cited within the body of this dissertation (these include Aktunc, 2012; Davis, 2013; Misra & Omorodion, 2011; Tabib, 2013; Tarhan & Yilmaz, 2014).

Recommendations

As a society we have moved way beyond the Industrial Age, where counting widgets measured progress and productivity. Society has moved into the Knowledge Age. A more effective mechanism for reporting and assessing software development

progress is needed to support the reduction of project cost overruns, failures, cancellations, and more significantly, a more responsible and effective use of taxpayers' dollars. Consequently, three recommendations emerge from this study.

The first recommendation relates to the population. The initial proposal for this research put forth a triangulation of the perspectives of the three basic sectors of government software-intensive development efforts: contractor, government, and Independent Verification and Validation (IV&V) contractors. Both the government and IV&V sectors declined to participate. Therefore, my first recommendation is to replicate this study with representatives from the government and their associated IV&V contractors. A researcher from each of those two groups could potentially conduct this research with less resistance.

A second recommendation is to research the viability of defining and implementing semantic metrics, based on knowledge, to provide more meaningful software completion status. Knowledge is independent of programming language, tools, methodology, and management process used. With respect to research into measurement based on knowledge, the framework for this future research would be focused on previous research on the topic of syntactic versus semantic software metrics and the use of Natural Language Processing (NLP) to capture and maintain semantic metrics (Govindarajan, 2004; Stein, 2004; Stein et al., 2009).

There is a marked difference between syntactic and semantic metrics. With syntactic metrics progress is measured using counts and calculations based on data

extracted from program code. Previous research indicated that semantic metrics, based on knowledge, was an alternative approach to assessing software quality (Govindarajan, 2004; Stein, 2004). Future research would look to expand that work and analyze semantic metrics with respect to software development progress and status. Additionally, a GAO report, specifically GAO-08-619, indicated that an investigation into a knowledge-based approach to funding might lead to an improvement of the current DOD funding processes (USDOD, 2008).

Semantic metrics are based on knowledge of the problem space, or the domain of the software application under construction (Govindarajan, 2004; Stein, 2004; Stein et al., 2009). Semantic metrics are based on analysis of a knowledge base of domain information that can be compared to data mined from a software solution. Additionally, because semantic metrics can be acquired from diagrams and documents, not just code, semantic metrics are available earlier in the SDLC than syntactic metrics enabling earlier detection of issues with a corresponding reduced cost to fix errors (Govindarajan, 2004; Stein, 2004).

Lastly, semantic metrics may provide (a) the basis for a semantic metrics suite that is an integral component of the software development process and require little extra effort outside of the software development process to capture and report, (b) the required management insight into a software development effort with respect to both progress and completion status, and (c) a bridge between the current gap of abstract guidelines at a macrolevel and syntactic metrics at the microlevel.

Implications

The implication for an individual manager, organization, and the community at large is that this study highlights the fact that the data required to assess and report software status does not exist. Software is a combination of art and science. Measurements applicable to the scientific partition of software development are defined and measurable, but ambiguous. The artistic partition of software development is less concrete, supported by guidelines, not definitive measures. Software development is a dynamic and unpredictable activity, not linear. Measures that are in many cases subjective, not objective, are subject to controversy. Measures that are not subjective, but rather objective, have multiple definitions and/or operationalizations. Discussions, or conversations, are difficult when based on such a weak foundation.

An approach to status meetings, with this understanding in mind from the outset, may facilitate less stressful meetings. This understanding of software progress, status, and measurement is critical to establishing a line of communication between procurers, producers, and oversight organizations. Progress and status are not equivalent words. Combined with the fluid nature of software development this distinction provides the cornerstone for improved communication between multiple parties when real status is not available. This communication gap can be further enhanced with relevant semantic metrics based on knowledge. Each application defines its own ontology that forms the basis for all software development efforts, regardless of application type, country of origin, programming language, and programmer efficiency. A global standard based on

ontology-based measures has the potential to unite procurers and producers and provide a commonality with respect to that which is currently missing; a relevant software status mechanism.

Conclusion

The issues surrounding software cost and schedule overruns continue. The U.S. government recently awarded a contract amounting to $121M to Accenture to fix and maintain the implementation of the website, www.healthcare.gov, that supports the Affordable Care Act, (Accenture, 2014; Howell & Dinan, 2014). This figure, $121M, exceeds the CGI Federal initial development cost of $93.7M in 2011 (Howell & Dinan, 2014) but does not include overruns prior to April 2014. This overrun is a most recent example of the lack of necessary mechanisms to provide insight into software development status and serves as a reminder of why this research was needed.

Multiple sources have expended significant effort to provide standards and guidance with respect to measures, metrics, and processes to support and improve software development efforts. Unfortunately, focus on assessing software development status is lacking. The focus of software metrics is on elements that can be objectively measured (e.g., counted, calculated), yet many elements of software products are subjective (e.g., quality of a requirement or design) and less amenable to measurement. Software development is a combination of art and science and iteration between them continues throughout a development effort. Until the industry devises a way forward to measure a moving and fluid target, obtaining a realistic measure of status is simply not

yet possible. Software is not done until it is done and currently taxpayer dollars are allocated to software metrics that are not relevant to assessing and reporting software development completion status.

References

Abdellatief, M., Sultan, A .B. M., Ghani, A. A. A., & Jabar, M. A. (2013). A mapping study to investigate component-based software system metrics. *Journal of Systems and Software, 86*, 587-603. doi:10.1016/j.jss.2012.10.001

Abran, A. (2010). *Software metrics and software metrology.* Hoboken, NJ: John Wiley & Sons.

Abreu, F. B. (1995, August). *Design metrics for object-oriented software systems.* Workshop presented at 9th European Conference on Object-Oriented Programming (ECOOP '95) Workshop on Metrics, Aarhus, Denmark. Retrieved from http://ctp.di.fct.unl.pt/QUASAR/ECOOP95/ecoop95_submission.pdf

Abreu, F. B., & Carapuça, R. (1994). Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software, 26*(1), 87-96. doi:10.1016/0164-1212(94)90099-X

Abreu, F. B., & Melo, W. (1996, March). Evaluating the impact of object-oriented design on software quality. *Proceedings of the 3rd International Software Metrics Symposium*, New York, NY, 90-99. doi:10.1109/METRIC.1996.492446

Accenture. (2014). Accenture Federal Services and the Centers for Medicare and Medicaid Services finalize agreement [Press release]. Retrieved from http://newsroom.accenture.com/news/accenture-federal-services-and-the-centers-for-medicare-and-medicaid-services-finalize-agreement.htm

Aktunc, O. (2012). Entropy metrics for agile development processes. *Proceedings of the 2012 IEEE 23rd International Symposium on Software Reliability Engineering*. Dallas, TX, 7-8. doi:10.1109/ISSREW.2012.36

Alhir, S. S. (1998). *UML in a nutshell: A desktop quick reference.* Sebastopol, CA: O'Reilly & Associates.

Archer, C., & Stinson, M. (1995). *Object-oriented software measures* (Technical Report CMU/SEI-95-TR-002). Unpublished manuscript, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu

Armour, P. G. (2004). Beware of counting SLOC. *Communications of the ACM, 47*(3), 21-24. doi:10.1145/971617.971635

Auberbach, C. F., & Silverstein, L. B. (2003). *Qualitative data: An introduction to coding and analysis.* New York, NY: New York University Press.

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering, 22*(10), 751-761. doi:10.1109/32.544352

Basili, V., Heidrich, J., Lindvall, M., Munch, J., Regardie, M., Rombach, D., … Trendowicz, A. (2007). *Bridging the gap between business strategy and software development.* Paper presented at 28th International Conference on Information Systems, Montreal, CA. Retrieved from http://www.cs.umd.edu/~basili /publications/proceedings/P124.pdf

Bazeley, P., & Jackson, K. (2013). *Qualitative data analysis with NVivo.* Thousand Oaks, CA: Sage.

Berry, D. M. (2008). The software engineering silver bullet conundrum. *IEEE Software, 25*(2), 18-19. doi:10.1109/MS.2008.51

Blaha, M. (2004). A copper bullet for software quality improvement. *Computer, 37*(2), 21-25. doi:10.1109/MC.2004.1266291

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

Booch, G. (1996). *Object solutions: Managing the object-oriented project.* Menlo Park, CA: Addison-Wesley.

Booch, G., Maksimchuck, R. A., Engle, M. W., Young, B. J., Conallen, J., & Houston, K. A. (2007). *Object-oriented analysis and design with applications* (3rd ed.). Upper Saddle River, NJ: Addison-Wesley.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The unified modeling language user guide* (2nd ed.). Reading, MA: Addison-Wesley.

Bouwers, E., Visser, J., & Van Deursen, A. (2012). Getting what you measure. *Communications of the ACM, 55*(7), 54-59. doi:10.1145/2209249.2209266

Briand, L. C., Morasca, S., Basili, V. R. (1996). Property-based software engineering measurement. *IEEE Transactions on Software Engineering, 22*(1), 68-86. doi:10.1109/32.481535

Brooks, F. P. (1995). *The mythical man-month* (Anniversary ed.). Reading, MA: Addison-Wesley.

Cane, S., McCarthy, R., & Halawi, L. (2010). Ready for battle? A phenomenological study of military simulation systems. *Journal of Computer Information Systems, 50*(3), 33-40. Retrieved from http://www.iacis.org

Carnegie Mellon University Software Engineering Institute (1995). *The capability maturity model: Guidelines for improving the software process.* Unpublished manuscript, Boston: MA: Addison-Wesley.

Carnegie Mellon University Software Engineering Institute (2012). CMMI® *for SCAMPI*^{SM} *Class A Appraisal Results 2012 mid-year update.* Unpublished manuscript, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu

Chandrika, S. M., Babu, E. S., & Srikanth, N. (2011). Conceptual cohesion of classes in Object-Oriented systems. *International Journal of Computer Science and Telecommunications, 2*(4), 38-44. Retrieved from http://www.ijcst.org

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for Object-Oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476-493. doi:10.1109/32.295895

Chidamber, S. R., & Kemerer, C. F. (1995). Author's Reply. *IEEE Transactions of Software Engineering, 21*(3), 263-265. doi:10.1109/32.372153

Chidamber, S. R., Darcy, D. P., & Kemerer, C. F. (1998). Managerial use of metrics for Object-Oriented software: An exploratory analysis. *IEEE Transactions on Software Engineering, 24*(8), 629-639. doi:10.1109/32.707698

Churcher, N. I., & Shepperd, M. J. (1995). Comments on "A metrics suite for object-oriented design". *IEEE Transactions of Software Engineering, 21*(3), 263-265. doi:10.1109/32.372153

Cohen, M. Z., Kahn, D. L., & Steeves, R. H. (2000). *Hermeneutical phenomenological research: A practical guide for nurse researchers.* Thousand Oaks, CA: Sage.

Concas, G., Marchesi, M., Murgia, A., & Tonelli, R. (2010). An empirical study of social networks metrics in object-oriented Software. *Advances in Software Engineering*, 1-21. doi:10.1155/2010/729826

Creswell, J. W. (2007). *Qualitative inquiry & research design: Choosing among five approaches* (2nd ed.). Thousand Oaks, CA: Sage.

Davis, N. (2103). Driving quality improvement and reducing technical debt with the Definition of Done. In J. Guerrero (Ed.), *AGILE 2013 Conference* (pp. 164-168). doi:10.1109/AGILE.2013.21

Day, L. E. (2009). Software Metrics. *Journal of Quality Assurance Institute, 23*(3), 12-19. Retrieved from http://www.qaiusa.com

DeMarco, T. (1982). *Controlling software projects: Management measurement & estimation.* Englewood, NJ: Yourdon Press.

DeMarco, T. (1995a). *Why does software cost so much? And other puzzles of the Information Age.* New York, NY: Dorset House.

DeMarco, T. (1995b). Mad about measurement. In T. DeMarco (Ed.), *Why does software cost so much? And other puzzles of the Information Age* (pp. 13-44). New York, NY: Dorset House.

DeMarco, T. (1995c). Software development: State of the art vs. state of the practice. In T. DeMarco (Ed.), *Why does software cost so much? And other puzzles of the Information Age* (pp. 125-135). New York, NY: Dorset House.

DeMarco, T. (1995d). Management-aided software engineering. In T. DeMarco (Ed.), *Why does software cost so much? And other puzzles of the Information Age* (pp. 46-70). New York, NY: Dorset House.

DeMarco, T. (1997). *The deadline: A novel about project management.* New York, NY: Dorset House.

DeMarco, T., & Lister, T. (2003). *Waltzing with bears: Managing risks on software projects.* New York: NY. Dorset House.

Dev, H., & Awathi, R. (2012). A systematic study of requirements volatility during software development process. *International Journal of Computer Science Issues, 9*(2), 527-533. Retrieved from http://www.ijcsi.org

Etzkorn, L., Banisha, J., & Davis, C. (n.d.). *Design and code complexity metrics for OO classes.* Unpublished manuscript, Computer Science Department, University of Alabama, Huntsville, Alabama. Retrieved from http://www.cs.uah.edu/~letzkorn/joop2.pdf

Etzkorn, L., & Delugach, H. (2000, August). *Towards a metrics suite for object-oriented design.* Paper presented at 34th International Conference on Technology of Object-Oriented Languages and Systems - TOOLS 34, Santa Barbara, CA. doi:10.1109/TOOLS.2000.868960

Etzkorn, L. H., Gholston, S., & Hughes, W. E. (Jr.) (2002). A semantic entropy metric. *Journal of Software Maintenance and Evolution: Research and Practice, 14*(4), 293-310. doi:10.1002/smr.255

Eveleens, J. L., & Verhoef, C. (2010). The rise and fall of the CHAOS report figures. *IEEE Software, 27*(1), 30-36. doi:10.1109/MS.2009.154

Farid, W., & Mitropoulos, F. (2013, April). NORPLAN: Non-functional requirements planning for Agile processes. Paper presented at *IEEE 2013 SoutheasternCon,* Jacksonville, FL. doi:10.1109/SECON.2013.6567463

Fenton, N., & Pfleeger, S. L. (1997). *Software metrics: A rigorous and practical approach* (2nd ed.). Boston, MA: PWS Publishing.

Financial services and general government appropriations for 2012: Hearings before a Subcommittee on Appropriations, House of Representatives, 112th Cong. 1 (2011a) (testimony of Jo Ann Emerson, Chair).

Fraser, S., & Manci, D. (2008). No silver bullet: Software engineering reloaded. *IEEE Software, 25*(1), 91-94. doi:10.1109/MS.2008.14

Gall, C. S., Lukins, S., Etzkorn, L., Gholston, P., Farrington, P., Utley, D., … & Virani, S. (2008). Semantic software metrics computed from natural language design specifications. *IET Software, 2*(1), 17-26. doi:10.1049/iet-sen:20070109

Galorath, D. (2011). *296 Billion dollars in DOD cost overruns: 2009 GAO weapons systems assessments.* Retrieved from http://www.galorath.com/wp/296-billion-dollars-in-dod-cost-overruns-2009-gao-weapons-systems-assessments.php

Gandhi, P., & Bhatia, P. K. (2012). Analytical analysis of generic reusability: Weyuker's Properties. *International Journal of Computer Science Issues, 9*(2), 424-427. Retrieved from http://ieeexplore.ieee.org

Govindarajan, R. (2004). *An empirical evaluation of information theory-based software metrics in comparison to counting-based metrics: A case study approach* (Masters Thesis). Mississippi State University, Mississippi State, MS.

Halstead, M. H. (1977). *Elements of software science*. New York, NY: Elsevier North Holland.

Hegel, G. W. F. (2010). *The Science of Logic* (G. Di Giovani, Trans.). Cambridge, UK: Cambridge University Press.

Henderson-Sellers, B. (1996). *Object-Oriented metrics: Measures of complexity.* Upper Saddle River, NJ: Prentice-Hall PTR.

Hofbauer, J., Sanders, G., Ellman, J., & Morrow, D. (2011). *Defense-Industrial Initiatives Group, Center for Strategic & International Studies: Cost and time overruns for major defense acquisition programs*. Center for Strategic Studies: Washington, D.C. Retrieved from http://csis.org/files/publication/110517_DIIG_MDAP_overruns.pdf

Howell, T. Jr., & Dinan, S. (2014, April 29). Price of fixing, upgrading Obamacare website rises to $121 million. [Press release] *Washington Times*. Retrieved from http://www.washingtontimes.com/news/2014/apr/29/obamacare-website-fix-will-cost-feds-121-million/?page=all

Hull, E., Jackson, K., & Dick, J. (2011). *Requirements engineering* (3rd ed.). London, UK: Springer-Verlag.

Humphrey, W. S. (1987). *Characterizing the software process maturity framework* (Technical Report CMU/SEI-87-TR-11). Unpublished manuscript, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu

Humphrey, W. S. (1988). Characterizing the software process a maturity framework. *IEEE Software 5*(2), 73–79. doi:10.1109/52/2014

Humphrey, W. S. (1989). *Managing the software process*. USA: Addison-Wesley.

Institute of Electrical and Electronics Association Standards Association. (2014). *Software and Engineering Standards*. Retrieved from http://www.standards.ieee.org/findstds/standards/software_and_systems_engineering_all.html

Iqbal, S., & Khan, M. N. A. (2012). Yet another set of requirements metrics for software projects. *International Journal of Software Engineering and its Applications, 6*(1), 19-28. Retrieved from http://www.sersc.org/journals/IJSEIA/

Janesick, V. J. (2011). "*Stretching" exercises for qualitative researchers* (3rd ed.). Thousand Oaks, CA: Sage.

Joint Technical Committee ISO/IEC JTC 1 Subcommittee SC 7. (2007). *Systems and software engineering – Measurement process (ISO/IEC 15939:2007[E]).* Switzerland: International Standards Organization. Retrieved from http://www.ieeexplore.ieee.org

Joint Technical Committee ISO/IEC JTC 1 Subcommittee SC 7. (2011). *Software engineering – Guide to the software engineering body of knowledge* (ISO/IEC TR19759:2005[E]). Switzerland: International Standards Organization. Retrieved from http://www.ieeexplore.ieee.org

Joint Technical Committee ISO/IEC JTC 1, & Software & Systems Engineering Standards Committee. (2010). *Systems and software engineering – Vocabulary (ISO/IEC/IEEE 24765:2010[E]).* Switzerland: International Standards Organization. Retrieved from http://www.ieeexplore.ieee.org

Joint Technical Committee ISO/IEC JTC 1, & Software & Systems Engineering Standards Committee. (2011). *System and software engineering – Architecture description* (ISO/IEC/IEEE 42010:2011[E]). Switzerland: International Standards Organization. Retrieved from http://www.ieeexplore.ieee.org

Jones, C. (1995). Backfiring: Converting lines of code to function points. *Computer, 28(*11), 87-88. doi:10.1109/2.471193

Jones, C. (2004, October). Software project management practices: Failure versus success. *Crosstalk, Journal of Defense Engineering,* Retrieved from http://www.sts.hill.af.mil

Jones, C. (2008). *Applied software measurement: Global analysis of productivity and quality* (3rd ed.). New York, NY: McGraw Hill.

Jones, C. (2010). *Software engineering best practices: Lessons from successful projects in the top companies.* New York, NY: McGraw-Hill.

Jørgensen, M., & Molokken, 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

Kitchenham, B., Pfleeger, S. L., Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering, 21*(12), 929-944. doi:10.1109/32.489070

Laanti, M. (2008). *Implementing program model with agile principles in a large software development organization.* Proceeding of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2008), Finland, 1383-1391. doi:10.1109/COMPSAC.2008.116

Lenski, G. (1970). *Human societies: A macrolevel introduction to sociology*. New York, NY: McGraw.

Li, W. (2000). Software product metrics: Using them to quantify design and code quality. *IEEE Potentials, 18*(5), 24-27. doi:10.1109/45.807276

Li, W., & Henry, S. (1993). *Maintenance metrics for the object-oriented design paradigm.* Paper presented at the First International Software Metrics Symposium, Baltimore, MD, 52-60. doi:10.1109/METRIC.1993.263801

Lincoln, Y.S. & Guba, E.G. (1985). *Naturalistic inquiry.* Thousand Oaks, CA: Sage.

Lorenz, M., & Kidd, J. (1994). *Object-oriented software metrics*. Englewood Cliffs, NJ: Prentice-Hall.

Ma, Y., Wu, H., Ma, X., Jin, B., Huang, T., & Wei, J. (2011). Stable cohesion metrics for evolving ontologies. *Journal of Software Maintenance and Evolution: Research and Practice* 23, 343-359. doi:10.1002/smr.509

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, SE-2*(4), 308-320. doi:10.1109/TSE.1976.233837

McConnell, S. (2010). *Code Complete* (2nd ed.). Redmond, WA: Microsoft Press.

Meadows, D. (2008). *Thinking in systems: A primer.* White River Junction, VT: Chelsea Green Publishing.

Meneely, A., Smith, B., & Williams, L. (2012). Validating software metrics: A spectrum of philosophies. *ACM Transactions of Software Engineering and Methodology, 21*(4), 24:1-24:28. doi:0:1145/2377656.2377661

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 63*(2), 81–97. doi:10.1037/h0043158

Misra, S., & Omorodion, M. (2011). Survey on agile metrics and their inter-relationship with other traditional development metrics. *ACM SigSoft Software Engineering Notes, 36*(6), 1-3. doi:10.1145/2047414.2047431

Morozoff, E. P. (2010). Using a line-of-code metric to understand software rework. *IEEE Software, 27*(1), 72-77. doi:10.1109/MS.2009.160

Moustakas, C. (1994). *Phenomenological research methods.* Thousand Oaks, CA: Sage.

Mueller, G. E. (1958). The Hegel legend of "Thesis-Antithesis-Synthesis." *Journal of the History of Ideas, 19*(3), 411-414. doi:10.2307/2708045

Mulcahy, R. (2011). *PMP® Exam Prep* (7th ed.). Minnetonka, MN: RMC Publications, Inc.

Nachmias, C, & Nachmias, D. (2008). *Research methods in the social sciences* (7th ed.). New York, NY: Worth Publishers.

National Defense Industrial Association, Systems Engineering Division. (2010). *Top software engineering issues within Department of Defense and defense industry.* Retrieved from http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/NDIA%20Top%20SW%20Issues%202010%20Report%20v5a%20final.pdf

Nguyen, V., Deeds-Rubin, S., Tan, T., & Boehm, B. (2007). A SLOC counting standard. Paper presented at the *22nd International Forum on COCOMO and Systems/Software Cost Modeling*, Los Angeles, CA. Retrieved from http://csse.usc.edu

Nominations before the Senate Armed Services Committee, Hearing before the Committee Armed Services, United States Senate, 112th Cong. 1 (2011a) (testimony of Senator Joe Lieberman).

Nominations before the Senate Armed Services Committee, Hearing before the Committee Armed Services, United States Senate, 112th Cong. 1 (2011b) (testimony of General Martin Dempsey).

Ott, L., Bieman, J. M., Kang, B-K., Mehra, B. (1995, June). Developing measures of class cohesion for object-oriented software. Paper presented at the *7th Annual Oregon Workshop on Software Metrics* (AOWSM'95), Portland, Oregon.

Park, R. E., SEI Software Metrics Working Group, & Software Process Measurement Project Team. (1992). *Software size measurement: A framework for counting source statements.* (Report No. CMU/SEI-92-TR-020). Unpublished manuscript, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu

Patton, M. Q. (2002). *Qualitative research & evaluation methods* (3rd ed.). Thousand Oaks, CA: Sage.

Paulk, M., Curtis, W., Chrissis, M. B., & Weber, C. V. (1993). *Capability maturity model for software* (Version 1.1) (Report No. CMU/SEI-93-TR-024). Unpublished manuscript, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu

Paulk, M., Weber, C. V., Garcia, S. M., Chrissis, M. B., & Bush, M. (1993). *Key practices of the capability maturity model* (Version 1.1) (Report No. CMU/SEI-93-TR-025). Unpublished manuscript, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu

Poels, G., & Dedene, G. (1997). *IEEE Transactions on Software Engineering, 23*(3), 190-195. doi:10.1109/32.585508

QSR International. (2013). *NVIVO 10 for Windows: Getting started.* [Unpublished manuscript. QSR International Pty Ltd. Retrieved from www.qsrinternational.com

Saldaña, J. (2013). *The coding manual for qualitative researchers* (2nd ed.). Thousand Oaks, CA: Sage.

Sandelowski, M. (1995). Sample size in qualitative research. *Research in Nursing and Health, 18*(2), 179-183. Retrieved from http://onlinelibrary.wiley.com/journal/10.1002/(ISSN)1098-240X

Schofield, J. (2005, April). The statistically unreliable nature of lines of code. *Crosstalk Journal of Defense Software Engineering.* 29-33. Retrieved from http://www.crosstalkonline.org

Schneidewind, N. F. (1992). Methodology for validating software metrics. *IEEE Transactions of Software Engineering, 18*, 410-422. doi:10.1109/32.135774

Senge, P. (1990). *The fifth discipline: The art & practice of the learning organization*. New York, NY: Doubleday.

Shenton, A. K. (2004). Strategies for ensuring trustworthiness in qualitative research projects. *Education for information, 22*(2), 63-75. Retrieved from http://www.iospress.nl/journal/education-for-information

Singh, G., Singh, D., & Singh, V. (2011). A study of software metrics. *IJECM International Journal of Computational Engineering and Management, 11*, 22-27. Retrieved from http://www.ijcem.org/

Software Engineering Standards Committee. (2005). *IEEE Standard for a software quality metrics methodology.* (IEEE Std 1061™-1998(R2009)). New York: Institute of Electrical and Electronics Engineers. Retrieved from http://www.ieeexplore.ieee.org

Software Engineering Standards Committee. (2006). *IEEE dictionary of measures of the software aspect of dependability.* (IEEE Std 982.1™-2005). New York:. Institute of Electrical and Electronics Engineers. Retrieved from http://www.ieeexplore.ieee.org

Software & Systems Engineering Standards Committee. (2008). *IEEE standard for software reviews and audits.* (IEEE Std 1028™-2008). New York: Institute of Electrical and Electronics Engineers. Retrieved from http://www.ieeexplore.ieee.org

Standish Group (2004). *2004 Third Quarter Research Report*. Retrieved from http://www.standishgroup.com

The Standish Group. (1995). *Chaos*. Unpublished manuscript. Retrieved from http://www.projectsmart.co.uk/docs/chaos-report.pdf

The Standish Group. (2013). *The CHAOS manifesto*. Retrieved from http://www.versionone.com/assets/img/files/CHAOSMnifesto2013.pdf

Stein, C. (2004). *Semantic metrics for source code and design* (Doctoral Dissertation). The University of Alabama: Huntsville, AL.

Stein, C., Etzkorn, L., Gholston, S., Farrington, P., Utley, D., Cox, G., & Fortune, J. (2009). Semantic metrics: Metrics based on semantic aspects of software. *Applied Artificial Intelligence, 23*(1), 44-77. doi:10.1080/08839510802573574

Tabib, R. (2013). Need 4 speed: Leverage new metrics to boost your velocity without compromising quality. *Proceedings of The 2013 Agile Conference),* Nashville, TN, 117-120. doi:10.1109/AGILE.2013.32

Tarhan, A. & Yilmaz, S. G. (2014). Systematic analyses and comparison of development performance and product quality of incremental process and agile process. *Information and Science Technology, 56*, 477-494.

Texel, P. P. (2013). Measure, metric, and indicator: An Object-Oriented approach for consistency. Paper presented at *IEEE 2013 SoutheasternCon,* Jacksonville, Florida. doi:10.1109/SECON.2013.6567438

The White House, Office of the Press Secretary. (2009). *Memorandum for the heads of executive departments and agencies* [Press release]. Retrieved from http://www.whitehouse.gov/the_press_office/Memorandum-for-the-Heads-of-Executive-Departments-and-Agencies-Subject-Government

Tigari, V. (2012). Information technology policies and procedures against unstructured data: A phenomenological study of information professionals. *Academy of Information and Management Sciences Journal, 15*(2), 87-106. Retrieved from http://www.alliedacademies.org

U.S. Department of Defense, Government Accountability Office. (2008). A knowledge-based funding approach could improve major weapon system program outcomes (Report GAO-08-619). Washington DC, Government Printing Office. Retrieved from http://www.gao.gov/products/GAO-08-619

U.S. Department of Defense, Government Accountability Office. (2009). *Defense acquisitions: Assessments of selected weapons programs* (Report GAO-09-326SP). Washington, DC: Government Printing Office. Retrieved from http://www.gao.gov/products/GAO-09-326SP

U.S. Department of Defense, Government Accountability Office. (2010). *Financial management improvement and audit readiness efforts continue to evolve* (Report GAO-10-1059T). Washington, DC: Government Printing Office. Retrieved from http://www.gao.gov/products/GAO-10-1059T

U.S. Department of Defense, Government Accountability Office. (2011). *Improved management insight* (Report GAO-11-53). Washington, DC: Government Printing Office. Retrieved from http://www.gao.gov/products/GAO-11-53

U.S. Department of Defense, Inspector General. (2012). *Enterprise resource planning systems schedule delays and reengineering weaknesses increase risks to DOD's auditability goals* (Report DODIG-2012-111). Washington, DC, Government Printing Office. Retrieved from http://www.gao.gov/products/GAO-10-1059T

Vago, S. (2004). *Social change* (5th ed.) Upper Saddle River, NJ: Prentice Hall.

Walker, W. (2007). *Ethical considerations in phenomenological research. Nurse Researcher, 14*(3), 36-45. Retrieved from http://nurseresearcher.rcnpublishing.co.uk

Weyuker, E. J. (1988). Evaluating software complexity measures. *IEEE Transactions on Software Engineering, 14*(9), 1357-1365. doi:10.1109/32.6178

Wheeldon, J., & Ahlberg, M. K. (2012). *Visualizing social science research: Maps, methods, and meaning.* Thousand Oaks, CA: Sage.

Whitfield, D. (2007). *Cost overruns, delays, and terminations: 105 outsourced public sector ICT projects* (ESSU Report No. 3). Adelaide, Australia: European Services Strategy Unit. Retrieved from European Services Strategy Unit website: http://www.european-services-strategy.org.uk/news/2007/ict-contract-chaos/105-ict-contracts.pdf

Winter, G. (2000). A comparative discussion of the notion of validity in qualitative and quantitative research. *The Qualitative Report, 4*(3&4). Retrieved from http://www.nova.edu/ssss/QR/QR4-3/winter.html

Yourdon, E. (2004). *Death march* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Appendix A: Current Software Measures and Metrics

Table A1 contains a summary of 111 metrics identified by seminal authors in the software metrics discipline. Twelve characteristics of software metrics are also identified. The 111 metrics are allocated to those characteristics, along with the reference(s) that introduced or identified a metric for that characteristic. The table does not include all seminal authors and all metrics, nor is it intended to. This study is not an exemplar of a grounded theory study. The table is a representative subset of the authors and their metrics that have influenced the software community.

The content of the table supports the fact that completion status is not addressed. The one characteristic in the table, progress, introduced by Lorenz and Kidd (1994), is associated with the metric number of contracts completed. Unfortunately the number of contracts completed is related to design elements, specifically classes. The number of contracts completed does not relate to the completion status of software functionality with respect to stated requirements completed.

Table A1

Allocation of Metrics to Software Characteristics

Metrics allocated to software characteristics			
Characteristic			References
Complexity	32	29%	Etzkorn, Banisha, & Davis, n.d.; Chidamber & Kemerer, 1994; Etzkorn & Delugach, 2000; Etzkorn, Gholston, & Hughes, 2002; Halstead, 1977; Henderson-Sellers, 1996; Li, 2000; Lorenz & Kidd, 1994; Ott, Bieman, Kang, & Mehra, 1995
Dependability	25	23%	Abreu, 1995, 1996; Abreu & Carapuça, 1994; Chidamber & Kemerer, 1994; Li, 2000; Lorenz & Kidd, 1994
Size	20	18%	Halstead, 1977; Li, 2000; Li & Henry, 1993; Lorenz & Kidd, 1994; Mozoroff, 2010
Coupling	7	6%	Chidamber & Kemerer, 1994; Lorenz & Kidd, 1994; Li, 2000; Li & Henry, 1993
Documentation	6	5%	Etzkorn, n.d.; Etzkorn & Delugach, 2000; Lorenz & Kidd, 1994
Cohesion	5	4%	Chidamber & Kemerer, 1994; Lorenz & Kidd, 1994; Etzkorn & Delugach, 2000
Level of effort	4	4%	Halstead, 1977; Lorenz & Kidd, 1994
Quality	4	4%	Lorenz & Kidd, 1994
Reuse		3%	Lorenz & Kidd, 1994; Henderson-Sellers, 1996; Gandhi & Bhatia, 2012
Faults	1	1%	Halstead, 1977
Knowledge	1	1%	Lorenz & Kidd, 1994
Progress	1	1%	Lorenz & Kidd, 1994
Total	*111*	*100%*	

Appendix B: Sample NoNotes.com Transcription

NoNotes.com provided the following transcript for participant P10.

P10: Hi, this is P10.

Interviewer: Hi P10.

P10: Hi, how are you?

Interviewer: Swamped.

P10: [Laughs] Well good, that's a good thing, right?

Interviewer: It is a good thing; it's a very good thing. I keep a lot of notes.

P10: There you go.

Interviewer: Hi, have you got a good hour?

P10: Sure.

Interviewer: Okay, from previous experience I would ask that you stay focused on what I'm really looking for which is capturing or accessing where the software is and reporting that information internally or externally. So if I interrupt it will be because I want to bring you back on track.

P10: Okay.

Interviewer: So just to run through . . . your informed consent has been provided to me and your participation in this interview is also interpreted as your consent. Is that okay?

P10: Yes.

Interviewer: Your privacy is protected. I'm taking notes and the only identification on this paper is your ID number which is P10. If by any chance any information creeps in that is either individual, organizational, project or geographical in nature information I will delete that from the transcript before you receive it. And any of that data if it's hard copy and needs to be maintained will be maintained in a locked container. So you are okay with that?

P10: Yes.

Interviewer: Number 4 - I hope that this interview will be approximately an hour and hopefully you can be interrupt free at that time?

P10: No problem.

Interviewer: I know you will do the best you can.

P10: Yeah I'll do my best, I mean if someone pops in I'll let you know that someone is here and I'll sort of wave them off. But it shouldn't be a problem, I don't have anything scheduled and this should be fine.

Interviewer: Okay and just to reiterate do I have your permission to record this interview given that its purpose is simply to get a file that represents the transcribed content and when I'm done with the original audio file I will delete it from the server. Is that okay?

P10: Yeah, that's fine.

Interviewer: Thank you. Okay I want to place emphasis in number 6, I have had experience with a couple of interviews and people tend to burst off which is fine - but this is focused on assessing where software is and reporting where the software is to superiors; either internally or externally. You can stop the interview at any time you like, okay?

P10: Okay.

Interviewer: Do you have any questions for me?

P10: Not at this time.

Interviewer: Okay, then part one, what positive experiences have you had with management mechanisms that you are most familiar with that include Gantt charts, risk matrix, any other management mechanism, and their ability to capture software completion status that gives you the ability to assess it? That is my first question to you.

P10: Okay, since this is the first one I guess I'll sort of go slow and make sure I'm on the mark. Positive experiences I guess that aspect of it, I've had experiences in terms of them being positive or not I'm going to talk about that and relay your questions I think; if that makes sense?

Interviewer: Okay.

P10: The things that we have looked at in the past and involved use cases, build matrices, etc., and basically I'm going to list the status points items that the software team had briefed to management internally and also externally. There really is not much difference (Status is status, I don't differentiate between internal and external when it comes to status, I try to be as transparent as I can with the customer at all times. But the use cases, build matrices, the Bugzilla entry status, the software lines of code or the SLOC count status of various points during the development and several others, you know them better than I do and those are the ones that came to mind when I was going through this in preparation for this meeting.

In terms of positive experiences, I thought all of those the items briefed by the software team were provided, they provided a reasonable amount of insight for the management team. In terms of it being positive, they were all positive, I mean . . .

Interviewer: Do you feel that your clients leave a status meeting with a clear feeling of where the software is, how much has actually been built and will the software really be delivered on schedule?

P10: Well in my experience and with what we went through here I would say initially the answer to that would be no and then over the course of time and working with the customer and the "oversight committee" I'll call them, we finally arrived at the a point where the information in terms of status and completion did satisfy their needs. I think initially it wasn't clear what level they were looking for and then through the course of working with the individuals and the others they kept asking for things and at some point in time they were satisfied.

And now -Their status provided to the customer was sort of developed during the course of the project. and that leads to my answer in the other questions, I'm saying that that was my experience, it was initially they weren't getting enough status it and then later they were after the team established what specifically the customer was looking for.

Interviewer: So they got educated along the way?

P10: True, true to that Yes.

Interviewer: Okay, moving to number two, what challenges did you deal with when dealing with the various ??? let's go back to question 1 before we hit question 2, I have another question for you. How you spoke to what positive experiences have you had with Gantt charts as a vehicle for measuring progress?

P10: I would say given that that is typically our only tool, if you mention a Gantt chart then it is what it is essentially and so it's a good tool in terms of the amount of insight you can do a lot with that gain. You can see the overall project, you can see multiple levels of activities, you can see activity linking is very important when it comes to those types of Gantt charts, and with you can see what needs to be done for first, second, third and that sort of thing, dependencies and activities and all that their relationships.

So I would call it a pretty good tool, I don't know how else we would do it I guess would be my response. I'm sure there are other ways to do it plan and execute a project.

Interviewer: No I hear you but do you think it reflects true progress?

P10: To an extent it does, I always know that when someone is reporting there are always things that they are not reporting, that risk hasn't necessarily been analyzed at their end yet so they feel they can deal with the problem whatever it is and still meet the current schedule. So I know there is always an undercurrent of things that are not being discussed that go with the tool if you will.

Interviewer: Okay, great. Let's move to 2, can you reflect on any real challenges you've had with the management mechanisms?

P10: Well the management mechanisms is are rigid I'll say and it they doesn't don't really allow for things like real world things events; for example like requirements creep, and the effect it can have in multiple unforeseen and unanticipated areas of a project. which we all know is something that you're always having to deal with in projects. One instance in particular has to do with SLOC, software lines of code, and how that affects CPU utilization. Requirements creep effects SLOC, and SLOC affects CPU Utilization, which then requires hardware engineering to re-asses the platform, which may cause

more changes in the software . . . and so on. There is no management mechanism that accounts for that in a waterfall development. So it really goes maybe a level up that comment. The only way to address this example is the to have a clear understanding of the requirements at the onset of that the project and the key/driving requirements in particular., things like CPU utilization which was a key requirement for us as we are developing. .

The tools didn't necessarily provide insight into you're monitoring SLOC completion and because you're monitoring SLOC is growth growing with requirements creep . . . consequently you never really know into what the final SLOC completion if you will, both ways and it's getting more and we are completing as we are going will be until the requirements are understood. SLOC is a moving target, so I don't think it's the best for providing status of completion.

But you are not seeing the link between SLOC for example and some key requirements like CPU utilization. As you add lines of code and as the new requirements are thrown out to you and it's growing your estimated SLOC numbers, how does that affect other things? The tool really didn't allow for that I would say, maybe we didn't know to ask for that at the time and it could have been provided . . . .

Interviewer: Do you feel that SLOC represents progress for completing requirements?

P10: To an extent it does but again the thing is you go through the project over time and you get new requirements understood or developed or thrown at you, I don't know how to say that because it's you who's doing this, you get halfway through and then all of a sudden you need to do this or the other that at the customers whim. I guess I would ask you: do you feel the requirements were set on the onset or did you feel like you got any new requirements as we went along?.

Interviewer: Oh there are always new requirements on every project, always.

P10: Yeah, so in general then to me I would sort of relate the SLOC count to the requirements; to get a new requirement your SLOC is most likely going to increase which is fine, you keep doing, going, you get more and more requirements and the software gets

bigger and more complex. But what's missing in the status tool is insight into how is that more requirements are ultimately tied to the new requirements other requirements, like CPU utilization? , and before you know it one new requirement brings down the whole project, like a house of cards. CPU utilization for us was a critical requirement in terms of turnover and acceptance. The status tools we used did not provide insight into the cause and effect of adding new requirements.

Interviewer: That is really a good point.

P10: Yeah, sort of the cross-matrix thing - domino effect, if you will. We see a lot of cost effect on the tools that we have, it was a rigid status of we did this, we planned this and this is where we are at. Okay that's good but what other concerns are there at this point, that kind of thing.

New Question. P10: When you use the term cross matrix thing are you referring to just interdependencies of SLOC with other software components or interdependencies of any 2 software components? e.g. SLOC and CPU, or SLOC and timing constraints, or document completion and bug fixes? That those interdependencies of all software components (not just SLOC) are not reflected in Gantt Charts or just in general

P10: The status tools we used did not provide any insight into interdependencies at all.

Interviewer:

Okay, question 3, again focusing on . . . well 3 focuses on positive and 4 focuses on challenges. What was positive about recording software status internally, was it easy to report it internally, was it difficult to report it internally, were you anxious, were you concerned? What thoughts go through your head when software status is reported to you both positive and challenging?

P10: Well I'd say for our experience, my experience that I had with software development and the software development life cycle it would basically be internally reporting. I didn't really have any stress necessarily. And certainly one thing that I remember that I carry forward to this day is the professionalism that the team demonstrated had while reporting it. And I say that because it's not clear that at the

proposal time, when we wrote the proposal for this stuff effort, that we did have bid enough time to provide the level of status that ended up being required.

When I say that and the software team got? Maybe they didn't do it but now they realize its requirement and they start to do it, that's a stress point for the team. All right they are doing stuff they didn't plan and lots of time status up to management isn't necessarily a favorite thing for the software development team to do. Although despite all of that they were very professional and did a very good job of providing status. So it was a very good experience in terms of internal and external actually because the status that came to myself, and others, was the same status, we basically briefed or the team briefed to the customers and all the stakeholders.

Interviewer: So you don't feel there were any challenges reporting externally?

P10: I think the only challenge again goes back to whether or not the hours were bid on the proposal to provide the breadth and depth of the status that we ended up providing. That's a challenge in terms of the cost kind of schedule and duration and time we had bid, it having to provide more status than originally thought stresses both of those. Because if you have a set team and now they have more things to do in terms of the software development status I would say that that is the stress and that's a challenge. I don't know if that is hitting the mark with what you're asking?

Interviewer: How about, how can I phrase this without bringing bias, did you feel that internal management and/or external management understood the status that was being presented? So they were comfortable leaving a conference room with a clear gut level feeling of where the software actually was and whether they thought you all could make it?

P10: Actually the team that we had, I have to say, didn't have a lot of previous experience with software development especially the level of software development we did accomplished, the amount of software we developed. So the experience probably wasn't there and when you talk of the internal briefings it was sort of a quick learn for our management, at least the non-software folks -that they had to do a SLOC - is the number

and the number is actual, So sort of in terms of a gut feeling, everybody internally management walked away satisfied that they had good insight into the development because of their lack of software development background (again, only the non-software managers).

But at the same time, the reason I say that is when we went external a lot of the external folks did have a lot of experience with software development. So they would have a better idea in terms of, I'll just give you an example. If typical software development status, 10 matrix and we were only [statusing] five and I'm just making something up here to make my point, that the external folks knew more than the internal necessarily, most likely in our case. Is that a fair thing to say, I mean I just call them like I see them?

Interviewer: Do you feel SLOC is a useful measure for progress for completion of requirements?

P10: I don't really; I mean software lines of code are only an artifact that should come out as the result of a requirement set. So again some simple numbers, if you had ten requirements you are going to have a predicable SLOC. Most likely I would say that the requirements are directly proportion to the lines of code. What's not seen or understood is requirements completion status, and how that will have an effect on all aspects of the software development. So simply stating SLOC completion doesn't give you the whole picture if at the last minute you get new requirements, obviously. I mean there is a correlation but I don 't see it as being the end of all the software status to whether you completed your line of code.

Interviewer: Well, we are now on 7 and when you sit back and reflect, what are your thoughts regarding the current contracting de facto mode of operation and what do you think about that. First what are your thoughts regarding what we do in the defense community as the standard and the impact it might have had on your experiences.

P10: Well on in my experience, the status that ended up being provided, if we call that the de facto, not being a software developer is and just being a technical lead in on the project, that saw that was adequate. A big concern that came out in the end was did we

really, and I keep going back to this I know, but did we really bid that effort to do the statusing at the level that we did. So not really having been through other software developments in my life, I've only done the de facto government type, I would say it was adequate but I'm sort of limited in my experience I would say.

Interviewer: Is there anything you can think of that you would have liked that you didn't have?

P10: A status tool that provides insight into the cause/effect of requirements creep. Just Also, in a general sense of the project, probably a little bit more work within the hardware software collaboration, I didn't see a lot of that, I saw one engineer made a design of a PC and sort of tossed it over the fence for the software folks and then they were done with it which in the end was okay but working a little more collaboratively with our work the hardware folks early on to make sure our platforms were adequate and met the requirements needs and whatnot. Also, given the fact that requirements creep is common, the hardware platform should accommodate a "predictable" amount of software growth.

So we did run into some snags on the hardware side in terms of that design. I don't know how much that affected you or not but for instance the original design had removable compact flash drives and that ended up being a poor designs in terms of security information assurance. So I don't know if that affected you in terms of going from a compact flash drive to removable hard drives if at all. But just in the general sense I think it would have been helpful if we had more hardware software collaboration.

Interviewer: You think that would have helped progress [crosstalk]

P10: It would have affected status, so I know you made that point earlier to make sure we stay on point. So in terms of hardware software collaboration I don't think there would be any benefits to the way that software development status was provided. But on the project level it would have solved a couple of problems if we had that early collaboration.

Interviewer: Number 8, do you have a different perception of reporting status with respect to software development, does your perception vary per software development life cycle phase?

P10: To an extent I guess it does. Early on after requirement development and then when the coding starts I sort of think that the matrix at that point are good in the best they can estimate but are not as relevant till you get a little further along in the development, like level two integration or you actually start to get towards the end of your builds, that kind of thing. I think early on it's not meaningless but it certainly has less meaning than it has towards the end just a general trend.

The reason I say that is early on I we don't know that if the requirements necessarily again are set and there are just too many unknowns early on; you don't know what problems you are going to have:, interfaces, and external companies that have to develop software for us, and just all the unknowns. So as you get farther along the risks seem to reduce.

Interviewer: If the measurement mechanisms that were used early on have less meaning then as we get closer to the end of the project then does that appoint to different quality and reporting accurate status? Is there a relationship there or?

P10: Yes, I would say there is, it's like the garbage in garbage out concept. If early on there is are unknowns, then the risk is high and I would just say that the numbers that are being reported early on there is carry more risk associated with those, that the data is actually not accurate. Whereas as you get closer to your software development end and you get closer to your final builds and you are starting a few tasks, at that point I would say the risk is very lower and your numbers are very increasing in accuracy at that point because the unknowns are gone..

Interviewer: Do you think there is one phase that the software development life cycle that stands out as the phase that you are most comfortable monitoring?

P10: If I was to pick a phase I would say level two integration, that whole phase if you will were more doing builds and we're going through at that point requirements are pretty

much concrete that any external software that's required is already completed and you are in the course of actually building the final product; I would say level two integration, where the hardware and software are integrated for system level testing.

Interviewer: Is there any one phase that stands out as the least relevant management mechanisms of monitoring?

P10: It would be again early so your requirements development, coding and unit test, level one unit, your unit level development I would guess; only again because of the potential risks and unknowns of that activity level during that phase.

Interviewer: Yeah, I hear you loud and clear; your message is coming through loud and clear so I personally thank you. When I ask if there is any one phase of SDLC that stands out in your line mind as being the worst to monitor I'm meaning like software requirements not system, software requirements, software design, software code, software unit test, software system test: do you think there is anyone of those phases that has mechanisms that are least relevant to monitoring progress and current status?

P10: I want to say the requirement phase only because there are so many unknowns when you are in that phase, you're working with the end user and it's hard to tell when you are going to be done. Does that make sense? The most painful point is probably during the requirement development and a close second being when you just start to get coding at the beginning so unit level testing, that sort of thing.

I only mentioned that because that phase and with my experience we had this external I think video card manufacturer I forget the name but they had to develop some software for us. So it was one of those you have an external contractor and you don't know if the parts are going to work or not and how that is going to affect your product and it becomes an interface issue and all that.

So as you reduce the unknowns going forward that is when things get easier; I guess it's a general trend.

Interviewer: Wow! Thank you, the best of three interviews I've had so far; thank you, you're on point on what you want to say and it's very clear. Again thank you. Is there anything you feel has been left unsaid?

P10: I said I did put my heart into it.

Interviewer: Yeah thank you. Is there anything you would like to ask me?

P10: Well, in terms of the subject matter or the next level up in general as far as …

Interviewer: Just in general.

P10: In terms of the government factor de facto, I'd be interested to know what other de factos are out there. I mean I'm sure there's hundreds and thousands of de facto going on software development schemes being implemented, I'm just curious more than anything. In the commercial world the process I assume is so much different than what we do in the DOD sector, not much different but still is I'll bet it's different, but probably more streamlined and probably I don't even know if I could say there is less metrics, some less reporting but probably not like anything we do.

Interviewer: I think it's more chaotic actually, it's more chaotic because there is not the heavy requirements for documentation, there is not that heavy requirement for process, the process is not so stringent and the emphasis is more on get it done, get it done, get it tested, get it done.

P10: So less control but more which doesn't help necessarily?

Interviewer: No, I think that is why agile software development has come from get this done mentality with short bursts of a month with less functionality to develop at one point in time. I think that is where agile has come from, the attempt to put some structure on commercial software. But anyway . . . .

P10: Yeah, they tend to deploy it early and then set up a support group and they deal with a reversioning metric, that is the how it goes.

Interviewer: Yeah, I just want to say thank you for your information, I should have the transcript in probably 48 hours or so. I'll take out any references that may have crept in that identify you as an individual, your organization, project information, and/o

geographical information. So all of that will be sliced before you get it back and hopefully when you get it you can really take a look at it and get it back to me as soon as possible.

Hopefully within 24 hours of getting it and if you can't I understand, just keep me posted.

P10: Well, with me if you send me the email I will . . . just be specific in what your needs are and in what your expectations are. Don't worry about . . . I'm just trying to get the job done so make sure you let me know and then I'll set my own priorities.

Interviewer: Yeah and please don't discuss your content with anybody else that you may know who is participating; let's wait till we are all done. Okay?

P10: [Okey dokey].

Interviewer: Thanks P10, most appreciative.

P10: No problem.

Interviewer: Have a great day.

P10: All right you too, thanks, bye.

Appendix C: Letter of Cooperation

<Community Research Partner Name>
<Contact Information>

<Date>

Dear <Researcher Name>,

Based on my review of your research proposal, I give permission for you to conduct the study entitled *Exploring Stakeholders Experiences Reporting and Assessing Software Completion Status* within the <Community Research Partner Name>. As part of this study, I authorize you to contact sample participants from the list of potential participants that I will provide and distribute to you, interview participants by phone, and distribute the results of your study to the participants at the completion of the study. Individuals' participation will be voluntary, at their own discretion, and on their own time. There will be no monitoring of their involvement by <Community Research Partner Name> unless an individual indicates directly to me that the involvement is affecting their job performance.

We understand that our organization's responsibilities include:

- Availability of each individual participant for a maximum total of 3-4 hours over a 6-8 week time period
- Private room with a telephone
- No interruption during the interview

I understand that there will be no mention of geographical location, company name, or participant identification in the data captured, data analysis, or final reports. I also understand that the researcher must adhere to the National Institute of Health (NIH) Protecting Human Research Participants (PHRP) requirements as well as the Walden University Institutional Review Board (IRB) requirements. I understand that the data collected will remain entirely confidential and may not be provided to anyone outside of the research team without permission from the Walden University IRB.

We reserve the right to withdraw from the study at any time if our circumstances change.

I confirm that I am authorized to approve research in this setting.

Sincerely,
<Authorization Official>
<Contact Information>

Appendix D: Consent Form

The following Consent Form, approved by the IRB, represents a modification of the Walden University Consent Form template. The modifications represent additions to the Walden Template that are relvant to this specific study.

CONSENT FORM

You are invited to take part in a research study exploring stakeholders' experiences using software metrics to capture software development completion status and report status internally within your organization and externally to stakeholders. The researcher is inviting stakeholders at least 25 years of age with at least 2 years of experience using metrics to detect and report software development completion status on software-intensive military applications and who are currently, or have been, involved in a defense application monitored by an IV&V contractor. You were selected because you satisfy these criteria. This consent form is part of a process called "informed consent" that allows you to understand this study before deciding whether to take part.

This study is being conducted by a researcher named Putnam P. Texel, a doctoral candidate at Walden University. You may already know the researcher as a previous Consultant but this study is separate from that role. The researcher is currently retired.

Background Information:
The purpose of this study is to explore stakeholders' experiences with software metrics and the ability to capture and report software development completion status as a first step in bridging the gap between theoretical identification and pragmatic application of software metrics.

Procedures:
If you agree to be in this study, you will be asked to spend no more than a total of 3-4 hours over a 6–8 week time period to:

- Read and return this Consent Form via email with the words "I consent" to the researcher at the email address below (30 minutes)
- Participate in an initial telephone interview (45-60 minutes)
- Review the transcription of the interview (45-60 minutes)
- Be available for up to 2 follow-up telephone calls to clarify data @15 minutes each (30 minutes)

Here are the issues that form the core of the interview:

- Describe your positive experiences with software metrics as a vehicle to capture software completion status?
- Describe your negative experiences with software metrics as a vehicle to capture software development status.?
- Describe any challenges you have experienced with software metrics as a vehicle to report software completion status internally within your organization?
- Describe any challenges you have experienced with software metrics as a vehicle to report software completion status externally to stakeholders?

Voluntary Nature of the Study:
Your participation in this study is voluntary. Everyone will respect your decision of whether or not you choose to participate in the study. No one at your place of employment will treat you differently if you decide not to be in the study. If you decide to join the study now, you can still change your mind later. You may stop at any time.

Risks and Benefits of Being in the Study:
Being in this type of study involves some risk of the minor discomfort that could be encountered in daily life, such as stress due to concern over anonymity. Confidentially is guaranteed by the procedures required by the National Institute of Health (NIH) Protection of the Rights of Human Participants (PRHP) and the Walden University Institutional Review Boards (IRB). The researcher's certification of completion of the NIH PRHP course is on file with the Walden IRB.

The benefit of the study is the improved focus on the gap that exists between management guidelines (e.g., Brook Law) and the counting metrics currently extracted from program code.

Payment:
There is no financial or material remuneration for participating in this study.

Privacy:
Any information you provide will be kept confidential. The researcher will not use your personal information for any purpose outside of this research project. The researcher will not include your name or anything else that could identify you in the study reports. Additionally, the researcher will not include any geographical, organization, or participant identification in the Dissertation or in any other published material related to this study. Data will be kept secure by several mechanisms including (a) storage of hard copy containing identification information in a lockable container, and (b) password-protection of electronic files that contain identification information. Data will be kept for a period of at least 5 years, as required by the Walden University.

Contacts and Questions:
You may ask any questions you have now. Or if you have questions later, you may contact the researcher via email at ptexel@gmail.com or cell phone (561) 346-4241. If you want to talk privately about your rights as a participant, you can call <name>. She is the Walden University representative who can discuss this with you. Her phone number is <number>. Walden University's approval number for this study is **<see Chapter 3>** and it expires on <date>.

Please keep this consent form for your records as well as a copy of the email indicating your consent.

Exiting the Study:
At the conclusion of the study, a debriefing letter will be submitted via email thanking you for your participation and requesting that your participation not be discussed with your family or peers.

Statement of Consent:

I have read the above information and I feel I understand the study well enough to make a decision about my involvement. By replying to this email with the words, "I consent", I understand that I am agreeing to the terms described above.

Appendix E: Participant Invitation Letter

<FROM>

<TO>

<RE:> Invitation to participate in research study

<First Name>,

As an organization we have been invited to participate in a doctoral research study. The purpose of the study is to explore stakeholders' experiences with software metrics and the reporting and assessment of software development completion status.

Your qualifications meet the criteria for inclusion in this study and you are invited to participate. It is expected that a total of approximately 3.0 – 4.0 hours of your time is required over a 6-8 week period. Your participation would require the following:

1) Read, sign, and return the attached Consent Form to the researcher (contact data below) (30 min)
2) Maintain a copy of the signed Consent Form for you records
3) Participate in a telephone interview conducted and recorded by the researcher (45 min)
4) Review the transcribed interview for accuracy (45 min)
5) Be available for up to two 15 minutes telephone calls to clarify (30 min)

Your privacy is guaranteed by the approval of the research study process by the Walden University Institutional Review Board (IRB). There is no financial or material remuneration for your participation however, a copy of the results of the research will be provided to you upon completion of the study.

Researcher contact data:

Name: Putnam P Texel
Ph.D. Candidate – Management: Specialization in Engineering
Walden University
Email: ptexel@gmail.com
Cell Phone: (561) 346-4241

Thank you in advance for your cooperation,

<Name>

Attached: Consent Form

Appendix F: Questionnaire

Research Questionnaire: Participant Number _____

#	Question	Response
1	How many years of experience do you have managing/assessing defense software applications?	0
2	Are you currently, or were you, involved with the development of a software application for a Government agency (e.g., DOD, non-DOD)?	** Select One **
3	Did an IV&V contractor monitor the project you are/were currently managing/monitoring?	** Select One **
4	What type of Agency are you currently supporting?	** Select One **
5	What is/was your role on the project?	** Select One **
6	What is the CMMI Level of your organization?	0
7	Did you report software completion status internally to your organization?	** Select One **
8	Did you report software completion status externally to stakeholders?	** Select One **

Appendix G: Interval Protocol

Date:

Time:

Place:

Interviewer:

Participant Number:

Before the Interview: There are a few items we need to cover before we start the interview.

1. Greeting and introductions.

2. Does your employer support you participating in this interview?

   YES NO

3. An Informed Consent form has been provided to you prior to this interview and you have signed and returned this form to me. Your participation in this interview is also interpreted as your consent. Is that your understanding as well?

   YES NO

4. Your privacy in this interview is guaranteed. You were provided a unique participant ID during the initial contact and only that number is used on all study documentation. A mapping of your participant ID and name is held in a locked container and/or a password-protected MS Word file. The responsibility to remember your number is solely yours. Is that your understanding as well?

   YES NO

5. The interview is intended to last approximately 45 minutes. Are you committed to being interrupt free for that time period?

   YES NO

6. Do I have your permission to record this interview? A transcript of the interview— recorded and transcribed, by a third party service, NoNotes.com— is provided to me as a .txt file that I will password-protect, then sanitize by removing any accidentally introduced identifying data. I will also delete the original audio file from the NoNotes.com server. The sanitized version will be forwarded to you for member-checking. Is this your understanding?

   YES NO

   The purpose of this interview is to explore your lived experiences when using software metrics to report and/or assess software development completion status. You are free to pass any question and you are free to terminate the interview at any time and/or withdraw from the study. Is this your understanding?

   YES NO

7. Do you have any questions for me?

   YES NO

Then let's begin.

The Interview:

Part I: Focus on capturing software development status

Question 1: What are your ***positive experiences*** with the metrics you are most familiar with and their ability to ***assess*** software completion status?

Question 2: What ***challenges*** have you faced with the metrics you are most familiar using and their ability to ***assess*** software completion status?

Question 3: Please address any personal "pain points" you have experienced in assessing software completion status.

Part II: Focus on reporting/assessing software development status

[Potential probe: SDLC Phase]

Question 4: What are your ***positive experiences*** when ***reporting, or assessing*** the reporting of, software completion status with software metrics ***internally***?
[Potential probe: emotional, physical, moral, thoughts]

Question 5: What ***challenges*** have you faced when ***reporting, or assessing the reporting of,*** software completion status with software metrics ***internally***? [Potential probe: emotional, physical, moral, thoughts]

Question 6: What are your ***positive experiences*** when **reporting*, or assessing the reporting of,*** software completion status with software metrics **e*xternally to stakeholders***? [Potential probe: emotional, physical, moral, thoughts]

Question 7: What ***challenges*** have you faced when ***reporting, or assessing the reporting of,*** software completion status with software metrics **e*xternally***? [Potential probe: emotional, physical, moral, thoughts]

Question 8: What are your thoughts regarding the defense contracting de facto mode of reporting and assessing software status and any impact it may have had on your experiences?

Question 9: What is your perception with respect to capturing/reporting/assessing software completion status and SDLC phase?

Question 10: Is there any one phase of the SDLC that stands out in your mind as the phase that has the most relevant metrics?

Question 11: Is there any one phase of the SDLC that stands out in your mind as the phase that has the least relevant metrics?

Question 12: Please address any personal "pain points" you have experienced in reporting software completion status.

After The Interview:

Thank you for your participation in this interview. A copy of the transcript will be forwarded to you for your approval. Please complete your review of the transcript within 24 hours of receipt. Is this timeline acceptable to you?

YES NO

Note: If the timeline is not acceptable, a mutually agreed timeline will be negotiated. Thank you again for your participation.

Figure G1. Interview protocol (figure created by P. Texel using Microsoft Word)

tions Mapped to Research Questions

Interview Questions Mapped to Research Questions	
ion	Interview question
e current software metrics supported the assessment of software ompletion status as perceived by stakeholders?	Q1, Q2, Q3, Q8
e current software metrics supported the reporting of software ompletion status as perceived by stakeholders?	Q4, Q5, Q6, Q7, Q8, Q12
stakeholders' experiences with software metrics and their relevancy evelopment Life Cycle (SDLC) phases?	Q8, Q9, Q10, Q11, Q12

Appendix H: Research Process Steps

In Table H1, the times entered into the column labeled Duration represent estimated time per participant unless preceded by the letter 'A' enclosed in parentheses, as in (A). The meaning ascribed to the asterisks in the column labeled location follows: (a) a single asterisk (*) indicates the activity will be completed at the researcher's home office, (b) a double asterisk (**) indicates that the activity will be completed by the organizational representative (either POC or participant) at any viable internet connection, and a triple asterisk (***) indicates that the activity will be completed by the Committee Chair, Committee Member, or URR representative at any viable internet connection

Research Process Steps				
Step #	Step	Duration	Exact location	Communication format
1	Researcher will establish contact with POC for each organization	1-hr	*	Telephone
2	Researcher will provide Proposal and unsigned Letter of Cooperation to POC	5-min	*	Email
3	POC will return signed Letter of Cooperation to me	5-min	**	Email
4	I will password-protect the signed Letter of Cooperation	1-min	*	N/A
5	I will request sampling frame from POC and provide criteria for inclusion	1-min	*	Email
6	POC will provide researcher with sampling frame	30-min	**	Email
7	Researcher will create a purposeful sample from all potential participants.	30-min	*	N/A
8	If POC contacts potential participants: Researcher will provide a Consent Form to POC to provide to each participant	5-min	*	Email
9	If Researcher contacts potential participants: researcher forwards Invitation Letter to potential participant	5-min	*	Email
10	If participant chooses to participate, researcher forwards Consent Form to participant	5-min	*	Email
11	In either case (Step 9 or Step 10) participant returns Consent Form with words “I agree” as text within the email	5-min	**	Email
12	Researcher will password-protect email, Consent Form, and Invitation Letter	5-min	*	N/A
13	Researcher will Create/Maintain Participant ID Mapping Schema	5-min	*	N/A
14	Researcher will password-protect Participant ID Mapping Schema	1-min	*	N/A

(table continues)

Step #	Step	Duration	Exact location	Communication format
15	Researcher select subset of three participants for Pilot Study	15-min	*	N/A
16	Researcher/Pilot Study participant agree interview date and time	15-min	*	Email
17	Researcher distributes Questionnaire and Interview Protocol to Pilot Study participant at least 24 hours in advance of date/time as agreed in Step 16	2-min	*	Email
18	I will conduct Interview with Pilot Study participant using CallRec.me	1-hr	*	Telephone
19	I will annotate hard copy of Interview Protocol as needed during interview (concurrent with Step 18)	0-min	*	N/A
20	I will stored annotated hard copy of Interview Protocol in locked container	2-min	*	N/A
21	CallRec.me forwards unsanitized transcription of Pilot Study participant interview (.txt file) to researcher	24-hr	CallRec.me server	Email
22	I will password-protect unsanitized transcript (.txt file)	2-min	*	N/A
23	I will delete original .txt file from the CallRec.me server	5-min	*	Internet
24	I will migrate unsanitized .txt file to a password-protected MS Word .docx file	2-min	*	N/A
25	I will migrate unsanitized .docx file to a sanitized MS Word .docx file	2-min	*	N/A
26	Password-protection will be removed from Word .docx file when a sanitized version of the transcription has been agreed.	2-min	*	N/A
27	Conduct member-check of sanitized Word .docx file with Pilot Study participant	30-min	*	Email
28	Researcher will create informal concept map of transcription content	30-min	*	N/A
29	Questionnaire and Interview Protocol will be updated	1-hr	*	N/A

(table continues)

Step #	Step	Duration	Exact location	Communication format
30	Researcher will contact remaining participants to participate in main research study	5-min	*	Email
31	Researcher/main study participant agree interview date and time	2-min	*	Email
32	Researcher distributes Questionnaire and Interview Protocol to main study participant at least 24 hours in advance of date/time as agreed in Step 31	1-hr	*	Telephone
33	I will conduct Interview with main study participant using CallRec.me	1-hr	* and **	N/A
34	I will annotate hard copy of Final Interview Protocol as needed during interview	2-min	*	N/A
35	I will store annotated hard copy of Interview Protocol in locked container	1-min	*	Email
36	CallRec.me forwards unsanitized transcription of main study participant interview (.txt file) to researcher	2-min	CallRec.me server	N/A
37	CallRec.me forwards unsanitized transcription of main study participant interview (.txt file) to researcher	24-hr	CallRec.me server	N/A
38	I will delete original .txt file from the CallRec.me server	2-min	*	N/A
39	I will migrate unsanitized .txt file to a password-protected MS Word .docx file	30-min	*	N/A
40	I will migrate password-protected unsanitized .docx file to a sanitized MS Word .docx file	2-min	*	N/A
41	Password-protection will be removed from Word .docx file when a sanitized version of the transcription has been agreed.	2-min	*	N/A
42	Researcher will create informal concept map of main study transcription content	2-min	*	Email

(table continues)

Step #	Step	Duration	Exact location	Communication format
43	Pilot study and main study participants indicate that the latest unsanitized transcription has been approved	2-min	* and **	Email
44	Create informal concept maps of main study participants' transcripts	15-min	*	N/A
45	Establish SPSS database with pilot and main study demographic data	60-min	*	N/A
46	Conduct distribution analysis using SPSS on participants' demographic data for inclusion in Chapter 4	60-min	*	N/A
47	Capture central tendencies using SPSS on appropriate demographic data (e.g., age, years of experience) for inclusion in Chapter 4	30-min	*	N/A
48	Establish NVivo database: Import sanitized MS Word transcription data into NVivo	(A) 2-hr	*	N/A
49	Conduct open coding of experiential data supported by NVivo	(A) 40-hr	*	N/A
50	Conduct axial coding of experiential data supported by NVivo	(A) 40-hr	*	N/A
51	Identify categories (collections of logically related codes)	(A) 40-hr	*	N/A
52	Identify themes (collections of logically related categories and relationships)	(A) 40-hr	*	N/A
53	Committee peer reviews evolving model	20-hr	* and ***	N/A
54	Iterate over steps 49-53 inclusive until no new codes/categories/themes are identified	40-hr	* and ***	N/A
55	Capture final summary textual and graphical data from NVivo (e.g., cluster map, concept map) for inclusion in Chapter 4	1-hr	*	N/A

Note. This table identifies CallRec.me as the third party vendor to record and transcribe interviews. CallRec.me went out of business before the study began. IRB granted permission to use NoNotes.com as the third party vendor. A Confidentiality Agreement was signed and submitted to the IRB. This table was not modified in order to keep the original steps as submitted to the IRB.

Appendix I: Example Concept Map: Participant P10

As indicated in Chapters 3 and 4, the content of each participant's interview formed the basis for the development a concept map and accompanying text summary. These three artifacts constituted the artifacts comprising the member-check process and were distributed to a participant for approval.

Figure I1 represents a sample concept map, specifically the concept map for participant P10 that I created from P10's interview transcript (see Appendix H). The concept map does not represent, nor is it intended to represent, all the detail contained in an interview transcript, but rather represents the key concepts that were included in the transcript, as well as any relationships between those concepts. The circle encloses the area of high traffic, the concept that has the most relationships, time.

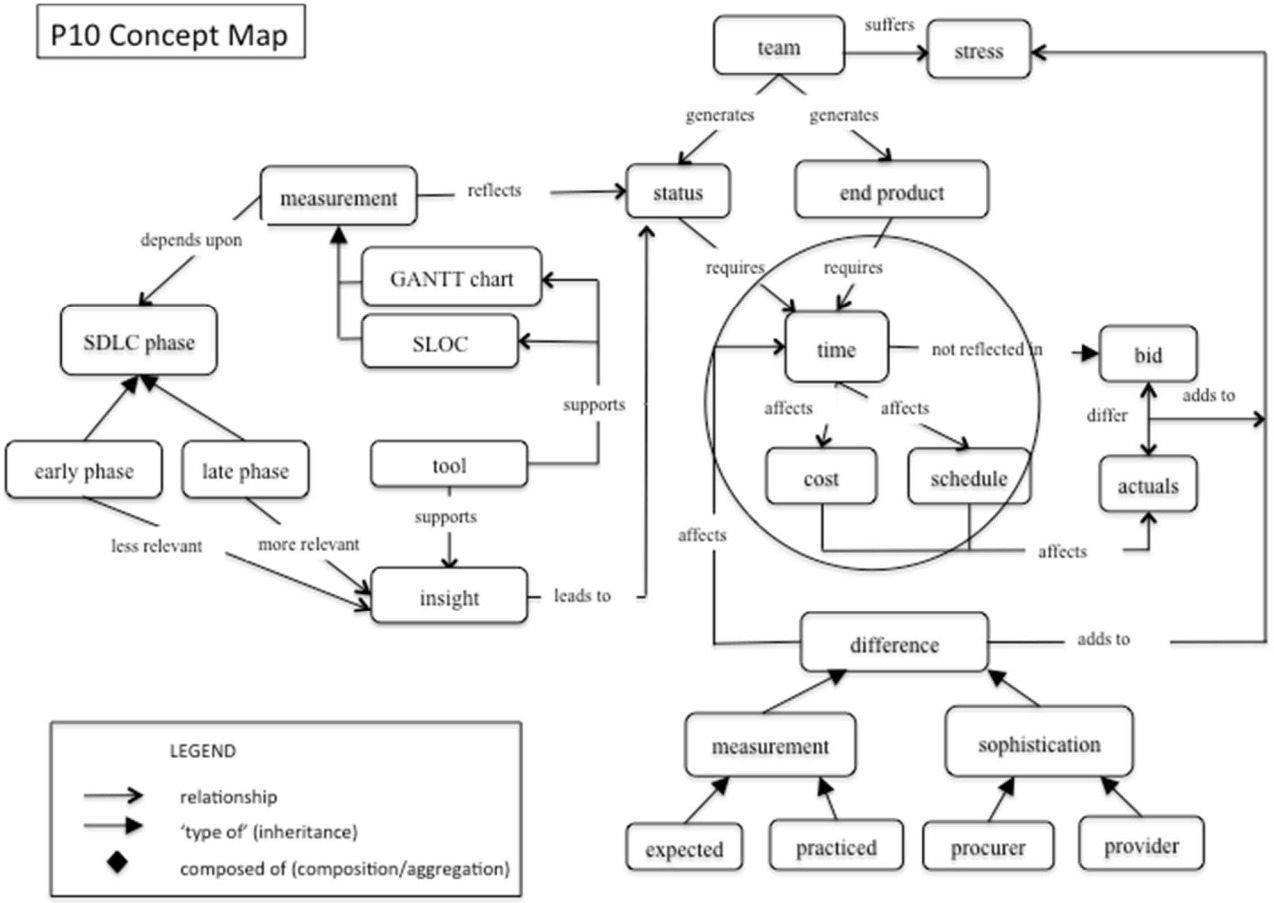


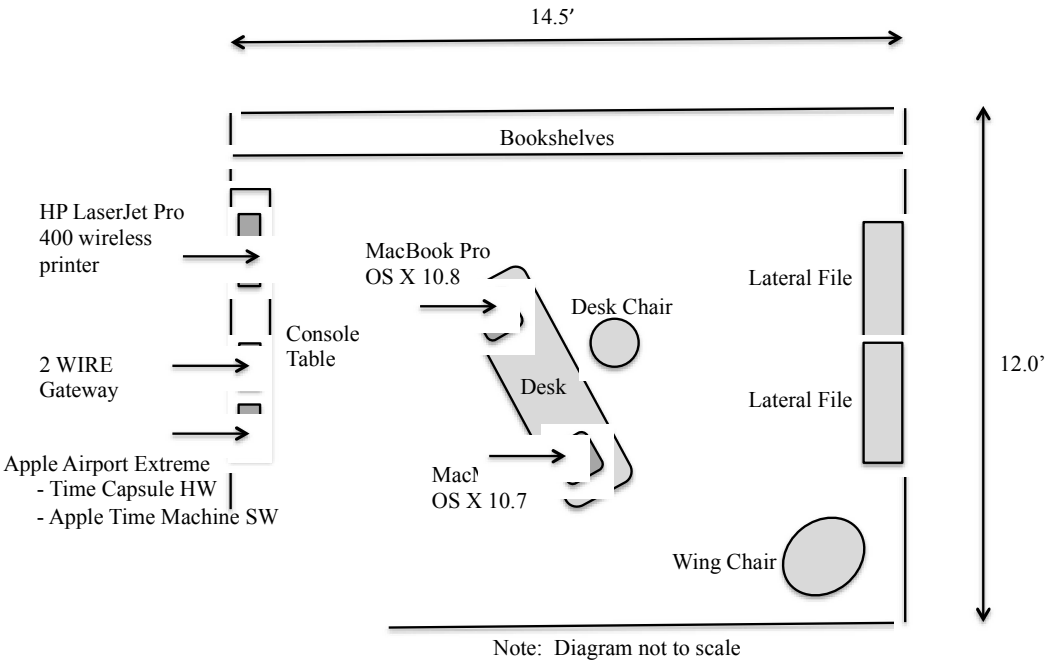
Figure I1. Example concept map: Participant P10.

Appendix J: Example Concept Map Summary: Participant P10

The following represents the textual summary, written to the concept map, for participant P10. The basis for the summary is the participant's approved transcript (see Appendix H) and the participant's approved Concept Map (see Appendix I).

> The effectiveness of measurements is dependent upon the specific phase of the SDLC being measured. Early on in the SDLC the measures are less relevant to insight/progress than they are later on in the SDLC. There are more unknowns early on in the project so the risk is higher and the measures may not reflect that. Gantt charts are useful but there is always an undercurrent of what is not being reported. SLOC is best when analyzed with other components of software development, for example CPU utilization and stringent spare requirements.
>
> The team suffers stress while supporting the generation and reporting of status while still under pressure to develop the software. Both of those efforts require time to complete and the actual time to capture and generate status is potentially not reflected in the bid. Consequently the actuals for cost and schedule increase resulting in differences between actuals and the bid. Differences also arise when measurements expected by the procurer are not those practiced by the developer. This requires time to resolve. The amount of time needed to resolve any differences depends upon the level of sophistication of the procurer and the developer. In the diagram notice the amount of activity around the concept of time (P10).

Appendix K: Home Office



Appendix L: Codebook

This codebook lists the eight demographic variables, in alphabetical order, included in the research study *Exploring Government Contractor Experiences Assessing and Reporting Software Completion Status*. Each variable in the list is introduced using the exact name, capitalization, data type, and measurement type used in SPSS and is followed by the associated question from the Questionnaire to which the data item corresponds. Following the research question, the response options and associated values are identified. These values and codes replicate the values and codes within the SPSS v21 model.

CMMI_lvl (constrained from 1 to 5)
Numeric
Nominal
Q6: What is the CMMI Level of your organization?

1 Initial
2 Repeatable
3 Defined
4 Managed
5 Optimizing
9 Prefer not to answer

gender
Numeric
Nominal
There is no specific question on the Questionnaire to capture this data point. The gender of a participant was intuited from the name and voice of the participant.

1 Male
2 Female
9 Other

gvt_Agency
Numeric
Nominal
Q2: Are you currently, or were you, involved with the development of a software application for a Government Agency (e.g., DOD, FAA, etc.)?

1 Yes
2 No
9 No answer

role
Numeric
Nominal
Q5: What is/was your role on the project?

1 Program Manager
2 Project Manager
9 Other

rpt_Int
Numeric
Nominal
Q7: Did you report software completion status internally to your organization?

1 Yes
2 No
9 No answer

rpt_Ext
Numeric
Nominal
Q8: Did you report software completion status externally to stakeholders?

1 Yes
2 No
9 No answer

type_Gvt_Agncy
Numeric
Nominal
Q4: What type of Government Agency are you currently supporting?
1 DOD
2 non-DOD
9 No answer

yrs_Exp
Numeric (constrained from 1 to 20)
Nominal
Q1: How many years of experience do you have assessing/managing defense software applications?

Appendix M: Interview Audit Trail

I maintained multiple audit trails throughout the study to organize and maintain participant status with respect to date/time of interviews and member-check of all research artifacts. Table M1 contains a sample audit trail that was maintained, specifically the audit trail of the date/time, start/stop times, and duration in minutes for each interview.

Please take note of the data points for entry16 in Table M1, the entry for Participant 20. This participant did not want to be interviewed by telephone and preferred email correspondence. I agreed to try and determine if the appropriate level of content could be provided using this media. The first round was not very successful. A follow-up email, including many follow-up questions, proved successful. I then combined data from both emails into the Interview Protocol. This in no way affected the quality/content of the study.

Table M1

Interview Audit Trail

#	Participant ID	Interview date	Start time	End time	Duration (minutes)
1	P1	3/14/14	15:03	15:46	43
2	P4	3/31/14	10:28	10:58	30
3	P5	4/9/14	11:30	12:19	49
4	P6	4/8/14	13:18	14:22	64
5	P7	4/2/14	10:00	10:32	32
6	P8	3/28/14	15:01	16:00	59
7	P9	3/10/14	16:11	17:01	50
8	P10	3/12/14	12:55	13:27	32
9	P11	4/9/14	14:15	15:27	72
10	P12	4/4/14	8:38	9:20	42
11	P13	4/3/14	11:55	12:31	36
12	P14	4/8/14	8:04	8:58	50
13	P15	4/7/14	13:00	13:53	53
14	P17	4/2/14	8:00	8:46	46
15	P19	4/3/14	15:00	15:36	36
16	P20	Participant requested email communication.			
17	P21	4/3/14	11:05	11:41	36
18	P22	4/7/14	8:30	9:04	34
19	P23	4/9/14	16:27	17:21	54
20	P24	4/1/14	15:47	16:29	42

Curriculum Vitae

Putnam P. Texel

EDUCATION

Ph.D. Candidate: Engineering Management, Walden University
(Expected completion February 2015)
M.S. (Magna Cum Laude) Mathematics Fairleigh Dickinson University. October 1968.
(Completed under Honors Research Teaching Fellowship)
B.A. (Cum Laude) Mathematics Fairleigh Dickinson University. June 1967.

Numerous computer related industry seminars (e.g. Ada, Java, J2EE, JSP, UML, C++, VxWorks, AdaTEST95)

AWARDS

2013. Golden Key International Honor Society
2006. Award of Special Recognition. U.S. Air Force Spacelift Range System Contract.
2004. Adjunct Faculty Member of the Year Award. Keiser College.
1998. Certificate of Appreciation. Joint Simulation System Joint Program Office.
1987. Recognition of Service Award. The Northwest Inland Empire Chapter of the Association of Computing Machinery (ACM).
1985. ACM Outstanding Service Award. Princeton Chapter of the Association of Computing Machinery (ACM).
1980. Employee Excellence Award. Control Data Corporation.

SUMMARY

An agent of change from functional software development processes and products to Object-Oriented software development processes and products based on the Ada (83, 95 and 2005) programming language for U.S. military services and their contractors. Projects included MIS as well as real-time embedded applications. As an agent of change one must be able to lead early adaptors as well as perform any task on the project: maintaining MS Project schedules and hiring/firing staff to creating design/code, writing/reviewing software specific documents, and/or implementing/running test artifacts. As an agent of change one must ensure the success of the first effort of a client by focusing on the staff, the process, the methodology, the products to be produced, and the software development toolset. Negotiated and managed sub-contracts as required.

Provided tutorials and technical papers (see Page 4) nationally and internationally at various activities including but not limited to ACM SigAda, CASE Forum in Sydney Australia, NATO Transition to Ada in The Hague Netherlands, Ada-Europe in Edinburgh, Scotland and various activities in and around the UK. Former Chairperson of the ACM SigAda Education Committee and the Jersey Shore Local SigAda. Author of 3 texts (See Page 4).

TECHNICAL SKILLS

Languages Utilized: UML, Ada83, Ada95, Ada 2005, Java, JSP, JavaScript, JDBC, C++, HTML, CSS, XML, DTD, XSLT. Assembly

Tools Utilized: DOORS, IBM/Rational Rose, GNATPro, CVS, SVN, AdaTEST95, oXygen XML Editor, MS Project, MS Office, Bugzilla

EXPERIENCE

Putnam P Texel LLC (2005 to 2010), President.
Provided instruction in Ada83, Ada95 and Object-Oriented technology processes and products based on [TEX97] (See Page 4). Major client was ITT Systems Division, PAFB, FL.

Consultant to ITT Systems Division (2005-2007)
Project and Technical Lead for Antenna Designate 1 (AD1). AD1 is an antenna pointing system used by Range Operators to track launched objects on the Western Range. AD1 receives, parses and validates a 240 bit HSD frame (serial data) from multiple sites, manages site prioritization, converts HSD data to Az/El for transmission to the antenna, reports antenna positioning and status data to the operator and executes every 10Hz (0.1 second). AD1 supports pre-mission configuration and post-mission report generation. DOORS was utilized for managing requirements. IBM/Rational Rose provided the tool support for the UML representation of the domain analysis and software design. AD1, a distributed system, was designed and implemented in Ada95 using AdaCORE GnatPro for the number crunching functionality and Wind River Tilcon Graphics Suite for the development and deployment of the Operator interface in C. IPL's AdaTEST95 was utilized to test the software at both the unit and use case level. CVS was the CM tool of

choice while Bugzilla tracked and managed software defects/issues. Performed the liaison/reporting duties between ITT and their Client, the U.S. Air Force. In addition to performing as Technical Lead, designed and implemented code, mentored staff, wrote the Software Development Plan (SDP), the Software Design Description (SDD) and set the structure/format and wrote tests for the Software Test Description (STD) used for software system testing. Managed QA interface as well as interfaced with IV&V Contractor.

Consultant to ITT Systems Division (2007-2009)
Project and Technical Lead for Antenna Designate 2 (AD2). AD2 was a follow on contract to the successful completion of AD1. AD2, based on AD1, handled 240 bit HDD frames, various sized LSD frames (to accommodate orbiting objects), and CS5 timing data on the Eastern Range. The functionality of AD2 more than tripled the functionality of AD1 and again executed every 10Hz. XML technology was introduced on AD2 to maintain data. The oXygen XML Editor was used to create XML files as well as to determine whether the XML file is well-formed and valid (if applicable). CM was upgraded to SVN. Ada2005 Components were utilized. Helped pass the "baton" to an ITT staffer. The contract came to a successful conclusion in October 2009.

P. P. Texel & Company, Inc. (1984 to 2004), President.
Training Division: Designed and developed a curriculum of 5 management and 7 technical courses (for education in Object-Oriented Analysis (OOA), Object-Oriented Design (OOD) and Object-Based Programming (OOP) with Ada83 based on Information Mapping and Reusable Educational Components (REC). REC enabled basic components to be utilized for both Management and Technical Training with additional Components added for Technical Training. This approach ensured the same "message" was being delivered to both management and technical personnel as well as enabled rapid client requested courseware customization. Delivered the curriculum both nationally and internationally. The quality of the curriculum and its supporting courseware provided the foundation for the growth of the company, leading to the development of 3 divisions: Education & Training, Software Development and Consulting.

Software Development Division: Provided software engineering design and development services to various programs (e.g. V-22 AFT/OFT, A-12 OFT/MT, SICBM OFP, LSD-41 CLASS MPCSOT).

Consulting Division: Provided QA and IV&V support to such programs as V-22 AFT/OFT, ATACC, AFATDS, LSD-41 CLASS MPCSOT.

Clients included but were not limited to Boeing, British Aerospace, Flight Safety International, GTE, General Dynamics, Grumman, Harris Corporation, McDonnell Douglas, Motorola, NASA, NATO, N&P Building Society, Raytheon, Rockwell International, Siemens, TRW, US Air Force, US Army, U.S. Marines.

Instructional Experience

Adjunct Instructor. New England Institute of Technology (2004-2005). Provided instruction in e-Commerce client side (HTML, JavaScript) and server side (Java, JSP) design and implementation using XML for input and output data.

Adjunct Instructor. Kaiser University (2004-2005). Provided instruction in Intermediate Algebra and College Mathematics. Recipient of the Adjunct Faculty Member of the Year Award 2004.

Adjunct Instructor. New York Institute of Technology (1973-1975). Provided instruction in Calculus and Differential equations at Fort Monmouth, NJ.

Assistant Professor. Fairleigh Dickinson University (1969-1972). Retained as Assistant Professor upon completing M.S. Provided instruction in both Mathematics (e.g. Calculus, Differential Equations) and Computer Science (e.g. FORTRAN) courses.

PUBLICATIONS: BOOKS

[TEX97] Use Cases Combined With Booch/OMT/UML: Process & Products. (With C.B. Williams) Prentice Hall. 1997. [ISBN 0-13-727405-X]

Ada Supplement to Accompany Concepts in Data Structures & Software Development (G. Michael Schneider and Steven Brunell) West Publishing Company. 1991. [ISBN 0-314-82978-4]

Introduction to Ada Programming: Programming with Packages. Wadsworth Publishing Company. 1986. [ISBN 0-534-06348-9]

PUBLICATIONS: TECHNICAL PAPERS

"Measure, Metric, and Indicator: An Object-Oriented Approach for Consistency". Proceedings of the 2013 IEEE SoutheastCon. April 2013.

"Use Cases Combined With Booch/OMT/UML: Process & Products". Proceedings of the Rational Users Group (RUG) Conference. February 1997.

"Use Cases and Categories: The Basis for Object-Oriented Project Infrastructure". Proceedings of the ASEET Symposium. June 1996.

"Ada_Education := Design_Concepts '+' Ada_Constructs;" Proceedings of the ACM SIGCSE Symposium. 1992.

"Case Study: Object-Oriented Requirements Analysis (OORA) Applied to AFATDS". Proceedings of the First Technical Symposium of AMP. (With S. Levine). 1989.

"Software Engineering Applied to Curriculum Development". (with L. Blackmon). Ada: Managing the Transition. Proceedings of the Ada-Europe International Conference. 1986.

"The Real Issues in Ada Education". Proceedings of NATO's Transition to Ada. 1986.

"The U.S. Army Model Ada Training Curriculum". Proceedings of the 2nd Annual Conference on Ada Technology.

"The CECOM Summer Faculty Research Program". Proceedings of the 2nd Annual Conference on Ada Technology.

"Ada '+' Style := Reliability;" Control Data PSI Excerpts. (With T. Walsh). 1981.

"Comparing Designs: A Methodology for Teaching Software Engineering". Proceedings of the 2nd Annual ASEET Symposium. 1987.