EFFECTIVE SOFTWARE ENGINEERING LEADERSHIP FOR DEVELOPMENT PROGRAMS

by

Marsha Cagle West

A Dissertation Presented in Partial Fulfillment

of the Requirements for the Degree

Doctor of Management in Organizational Leadership

UNIVERSITY OF PHOENIX

November 2010

EFFECTIVE SOFTWARE ENGINEERING LEADERSHIP FOR DEVELOPMENT PROGRAMS

by

Marsha Cagle West

November 2010

Approved:

Johnny L. Morris, Ph.D., Mentor

Douglas M. LePelley, Ph.D., Committee Member

Joseph B. Baugh, Ph.D., Committee Member

Accepted and Signed: [illegible] 11/09/2010
Johnny L. Morris — Date

Accepted and Signed: [illegible] 11/09/2010
Douglas M. LePelley — Date

Accepted and Signed: [illegible] 11/09/2010
Joseph B. Baugh — Date

[illegible] 11/23/2010
Jeremy Moreland, Ph.D.
Dean, School of Advanced Studies
University of Phoenix — Date

ABSTRACT

Software is a critical component of systems ranging from simple consumer appliances to complex health, nuclear, and flight control systems. The development of quality, reliable, and effective software solutions requires the incorporation of effective software engineering processes and leadership. Processes, approaches, and methodologies for software engineering continue to emerge, yet software programs continue to experience high failure rates. This qualitative grounded theory study explored the software engineering and leadership practices applied to software development programs to identify successful and unsuccessful software development and leadership approaches. The problem investigated the leadership approaches for the management of software development programs continuing to result in increased cost, missed deadlines, reduced reliability, reduced quality, and failed programs. Data was collected from 71 participants actively involved in software development programs. The study resulted in development of a software engineering leadership theory focusing on the areas of environment, resources, and processes. The leadership environment must incorporate approaches for effective communication, fostering teamwork, empowerment, intelligence, and leadership by example. Resources for software engineering implement approaches for requirements management, process application, program planning, and system testing. Effective software development processes include incremental development, agile methodologies, CMM/CMMI philosophy, and waterfall development. The integration of environment, resources, and processes provides a theory for effective software development leadership resulting in improved product development and increased product quality.

DEDICATION

This dissertation is dedicated to the many members of my family who provided love, support, and encouragement throughout the process. The willingness of family members to listen when needed, support when required, and assure solitude when requested provided the environment needed to pursue a doctoral degree. I can never fully express my gratitude for the continual support during the journey and the encouragement to pursue any challenge.

ACKNOWLEDGMENTS

The challenge of graduate school presented the opportunity to learn and grow as a scholar, practitioner, and leader. The learning experience would not have been complete without the guidance and support of my mentor, Dr. Johnny Morris. I want to thank Dr. Morris for his guidance and direction during the journey. His mentorship, guidance, and encouragement provided a clear path for success. Thank you for creating a positive learning environment and providing valuable insight.

I would also like to thank my committee members, Dr. Douglas LePelley and Dr. Joseph Baugh. These individuals provided valuable insight in conducting the research study and developing the dissertation. Their eagerness to provide suggestions for improvement provided the incentive to strive harder and improve performance. Dr. LePelley and Dr. Baugh freely shared their knowledge and experience to improve my learning, research study, and dissertation.

Several leaders and co-workers provided support and encouragement for this effort. Dr. William Craig, Dr. Willie Fitzpatrick, and Mr. Fred Reed provided guidance and support on software engineering, leadership, and research. Mr. Jim Sacco, Ms. Sue McClung, and Ms. Joan Holt were always prepared with words of encouragement.

Finally I want to acknowledge family and friends who were always eager to assist and quick to encourage. Thank you to everyone who contributed to my education and growth. Thanks to each of you, the journey was rewarding and successful. Knowing such a tremendous and eager support system is available provides the courage to pursue the next challenge.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Software is present in many systems used everyday ranging from items such as home appliances and mobile telephones (Stackpole, 2008) to next-generation automobile navigation systems and flight control systems for the United States Space Shuttle (Schneidewind, 2007). The field of software engineering focuses on developing and implementing software programs for simple and complex applications (Sommerville, 2007). Software engineering involves the integration of the complex processes of software design, development, testing, and integration for software intensive systems (Pfleeger & Atlee, 2010; Sommerville).

Leadership of software development programs requires the application of development processes throughout the software engineering life cycle (Gefen, Zviran, & Elman, 2006). Even with the continued identification and implementation of development processes, software programs continue to experience failures resulting in increased costs and delayed product availability to consumers (Gottesdiener, 2008; Woolridge, Hale, Hale, & Sharpe, 2009). To improve software engineering efforts, leadership must identify and implement the most effective processes for the development life cycle (Tesch, Kloppenborg, & Frolick, 2007). This study explored leadership approaches for software engineering to identify processes for improvement, which may reduce project failures, cost overruns, and reliability issues in software development programs.

Chapter 1 provides background information on software engineering programs and the impact of failed development efforts. Determining leadership and development processes which contribute to successful project outcomes, processes which result in negative impacts, and processes which require improvement, may assist leadership in improving software engineering approaches for the project life cycle. Chapter 1 presents the research problem, research purpose, study significance, theoretical framework, assumptions, and limitations for the study.

Background of the Problem

As technology and computer applications continue to grow, expand, and improve, the demand for complex, reliable software applications also increases (Adams, 2008; Pressman, 2010; Sommerville, 2007). Software is an integral part of products used in everyday and advanced applications for consumers and industry (Basili et al., 2008; Probert, Hunt, Fraser, Fleury, & Holden, 2007). As Booch (2008) noted, “we as a professional community have developed technology that has changed the way individuals live, businesses operate, communities interact, and nations and civilizations thrive and expand” (p. 8). Failures in commercial software programs result in delayed availability, reduced quality, and increased cost to the consumer (Basili et al.; Rubinstein, 2007). Failures in Government and military software programs result in reduced capability, limited availability, and increased cost to the taxpayer (King, 2007).

The field of software engineering continues to grow and evolve (Pressman, 2010; Sommerville, 2007). The increased failure rate of software programs results in reduced quality products, increased costs, and delayed programs (Pressman; Sommerville). Software development initiatives often fail and completed projects are often more costly to produce than originally predicted (Adams, 2008). In the United States, annual expenditures on software development projects are approximately $275 billion with 70% of development efforts unsuccessful (Wallace & Keil, 2004). Software engineering programs often exceed budget and schedule as a result of the implemented development and leadership processes. Application of standardized software engineering processes to software development programs often leads to excessive cost overrun and program failures (Pino, Garcia, & Piattini, 2008; Sommerville, 2007).

The Standish Group report noted for projects completed, 46% resulted in time and cost overruns (Rubinstein, 2007). Telang and Wattal (2007) noted faulty software costs organizations in the United States approximately $60 billion per year. These increased costs, delayed availability, and reduced quality result in impacts to individuals, organizations, and society (Kirova, Kirby, Kothari, & Childress, 2008). These impacts range from minor inconveniences from failures of home appliances to errors in health care, transportation, aerospace, and nuclear industries which can result in injury or death (Adams, 2008; Kruchten, 2008).

Research has shown project leadership is a critical success factor of any software program encompassing organizational factors such as culture, strategy, and interaction (Tesch et al., 2007). Ineffective leadership processes increase the risk of project failure and the inability to reach organizational goals (Tesch et al.). Sapienza (2005) observed ineffective leadership for software development programs impacts schedule, cost, employee morale, and product quality. In the information technology (IT) environment, research revealed complex IT projects resulted in failure with only 16% of projects considered successful (Brown & McDermid, 2008). The most noted reason for failure of these projects related to leadership inability to implement best practices for software development (Brown & McDermid). Reducing the occurrences of failed or incomplete development efforts requires the application of leadership processes to meet functionality requirements while providing reliable and quality software on schedule and within budget (Agrawal & Chari, 2007).

Statement of the Problem

The general problem concerns the continuing high failure rate for software development programs (Sommerville, 2007; The Standish Group, 2009). The software engineering field encompasses a framework of paradigms, methodologies, and approaches to software development (Pressman, 2010; Sommerville). Even with the incorporation of these approaches, software development programs continue to have high failure rates affecting individual and business environments (Dalcher & Benediktsson, 2006). These software failures result in organizational cost overruns, increased schedule, and failure to meet consumer needs (Gefen et al., 2006).

Complex systems and common computer based products require software to function (Brown & McDermid, 2008; Sommerville, 2007). Organizations depend on successful software development programs to support computer applications (Hadar & Leron, 2008), military systems (King, 2007), and space program initiatives (Schneidewind, 2007). Consumers depend on computer products for everyday applications. Software is an integral part of the products used daily by consumers (Linden, Ortega, & Hong, 2010; Xu & Brinkkemper, 2007). Cellular phones, satellite television, automobile systems, and aircraft flight programs all rely on integrated software (Adams, 2008). The specific problem is current leadership approaches to the management of software development programs continue to result in increased cost, missed deadlines, reduced reliability, reduced quality, and failed programs (Cerpa & Verner, 2009; Dalcher & Benediktsson, 2006; Gottesdiener, 2008; Horn, 2009; Mizell &

Malone, 2007; Mukherjee, 2008; Pino et al., 2008; Sommerville; The Standish Group, 2009; Xu & Brinkkemper).

Purpose of the Study

The purpose of this qualitative grounded theory study investigated leadership and development practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. A qualitative study was appropriate to collect textual data from participants, ask broad general questions, and analyze these responses for themes in a subjective manner (Creswell, 2008; Shank, 2006). The goal of this study was to explore and analyze leadership and development processes for software engineering to identify a grounded theory of characteristics that result in successful software development programs. The research explored the experiences and unique perceptions of leaders and software developers actively involved in software development programs.

Theoretical sampling was used to select participants with experience leading software engineering efforts and developing software programs. Theoretical sampling provided participant selection relative to the investigation of the research question and purpose to provide insight for the developing theory (Bogdan & Biklen, 2007; Corbin & Strauss, 2008; Creswell, 2008; Leedy & Ormrod, 2005; Neuman, 2003; Shank, 2006). The researcher selected participants from the population with experience and backgrounds that provide knowledge and information to address the research purpose (McMillan & Schumacher, 2006).

The study invitation was provided to a research and development organization in Alabama with a population of 600 employees. The demographic questions were used to select the individuals with experience in software engineering and leadership. The number of participants was 12% of the invited population or 71 respondents.

This study employed a grounded theory design to examine processes and develop theories on which leadership practices and processes facilitate successful programs. Qualitative grounded theory concentrates on participant actions and interactions related to a topic of study to develop a theory on the processes and relationships (Corbin & Strauss, 2008; Creswell, 2008; Glaser & Strauss, 1967, 2007; Leedy & Ormrod, 2005). The grounded theory design was appropriate for the study to obtain information from software development leaders to identify trends, behaviors, and characteristics (Creswell; Glaser & Strauss) for successful management of software programs. Creswell noted grounded theory designs are appropriate when attempting to investigate and describe views, beliefs, attitudes, or aspects for a particular group.

The grounded theory research design focused on generating theory instead of proving theory (Glaser & Strauss, 2007) on leadership processes for improving software development initiatives. This study explored leadership approaches and software development methodologies to determine processes which are successful, processes which are negative impacts to performance, and processes which require update to enhance program development initiatives. The study focused on leaders and team members of software development programs at a research organization in Alabama.

Significance of the Problem

The software engineering field encompasses numerous methodologies, paradigms, and quality processes for use in development programs (Pressman, 2010; Sommerville, 2007). Application of these approaches does not always result in successful projects (Adams, 2008; Dalcher & Benediktsson, 2006; Gefen et al., 2006). Software development programs experience high failure rates resulting in cost overruns, increased schedule, reduced quality, and failure to meet consumer demands (Adams; Dalcher & Benediktsson; Gefen et al.). Investigation of the leadership and development approaches applied to software development programs could result in development of theories and approaches that are significant to leadership and the field of software engineering.

Significance of the Study

Due to the demand for improved software technologies, the number of software development projects started tend to double every 2 to 3 years, but the success rate for programs has not substantially improved (Tesch et al., 2007; The Standish Group, 2009). Software programs continue to experience large failure rates resulting in increased costs, reduced capabilities, and product delays to consumers and organizations (Dalcher & Benediktsson, 2006; Horn, 2009; Jones, 2008; Mizell & Malone, 2007; Rubinstein, 2007; Tesch et al.). The research study contributed to the body of software engineering information through the identification of processes, approaches, and practices which contribute to the increasing failure rates for software programs.

The grounded theory research design supported the development of theory for leadership processes based on the raw data collected on experiences and perceptions (Glaser & Strauss, 2007). Identification of effective and ineffective processes may lead to continued process improvement to enhance the success rate for development programs. The themes and trends identified in this study contribute to the development of theories and models (Creswell, 2008; Shank, 2006) for successful software development. The software engineering community may improve the state of the practice through the development of leadership and process theories which contribute to the success of software development programs. The identification of processes considered negative impacts to performance and project success may provide additional areas of research and opportunities for improvement.

Significance of the Study to Leadership

Leadership focuses on improving the performance of software engineering programs by applying the best applicable practices and procedures to maximize resources, maintain quality standards, and produce usable artifacts (Gefen et al., 2006). Leadership approaches and processes are critical success factors for software engineering and development programs (Tesch et al., 2007). To improve the success of software development programs, leadership must focus on identification of successful practices, removal of negative practices, and recognize opportunities to improve practices to maximize resources, maintain quality standards, and produce usable artifacts (Gefen et al.). Identification of successful practices and procedures are essential to achieving successful software development programs for leadership (Brown & McDermid, 2008).

This study was significant to software engineering leadership since the data collected was used to develop theories on leadership processes for successful software engineering. The numerous paradigms, development methodologies, and process improvement initiatives are not a straightforward solution to software development (Pressman, 2010; Sommerville, 2007). Individual development efforts select and tailor available approaches and methods to match goals and objectives. The changing software environment requires new and updated processes (Basili & Zelkowitz, 2007; Bell, 2008;

Glass, 2008; Sommerville) to meet the growing challenges for leadership in software development (Boehm & Valerdi, 2008).

This qualitative grounded theory study investigated the views and perspectives of software development leadership and experts. The data collected was analyzed to identify trends and themes on leadership characteristics and development processes that result in successful and unsuccessful programs. The data collected contributed to the development of theories and methodologies for enhancing leadership performance during software development efforts. The identification of successful and unsuccessful leadership approaches for software development activities may contribute to the improvement in leadership approaches and provided additional insight into appropriate methods to integrate into software engineering activities for success. Results of the study contributed to the software engineering body of knowledge to expand the available information on leadership approaches and development processes that should be applied to improve software development program initiatives.

Nature of the Study

The research problem, research question, and research goals determine the research method and design to be used to obtain data relevant to the topic of study (Creswell, 2008; Glaser & Strauss, 2007). The nature of this study focused on obtaining views, observations, and perspectives from a specific population with experience in software engineering. The following sections provide a discussion of the selection of a qualitative grounded theory research design as an appropriate framework to study leadership attributes and processes that lead to successful software development programs.

Overview of the Research Method

This qualitative study used a grounded theory design to investigate the leadership process and practices applied to software development programs to explore and understand the approaches that lead to program success. The qualitative study supported collecting observations and results on the application of various software development processes from leadership and team members. The data collected from study participants was obtained by developing general open-ended questions on software engineering methodologies, approaches, and processes. The study responses were analyzed to identify trends, themes, characteristics, and behaviors which support successful programs.

A qualitative research methodology was appropriate to this study to explore leadership processes and characteristics that contribute to successful software engineering programs. The intent of this study was not to measure specific dependent and independent variables related to program failures as is performed in quantitative research (Creswell, 2008). The intent of this study was to generate a theory of leadership processes that contribute to successful software development initiatives based on experiences and observations (Bogdan & Biklen, 2007).

Overview of the Design Appropriateness

This qualitative research study used a grounded theory design to explore the experiences and perceptions of software engineering leadership. The grounded theory design was appropriate for the qualitative data collection to pose general, broad questions to participants to obtain unrestricted perspectives (Creswell, 2008). The grounded theory design allowed respondents to provide thoughts, views, and perspectives (Corbin & Strauss, 2008) on software engineering and leadership processes. The results obtained were reviewed to identify trends, themes, and concepts on successful and unsuccessful software engineering processes.

Research Questions

This qualitative grounded theory research study focused on leadership practices for software engineering programs. The research question focused on the investigation of leadership impacts for program development. Research question 1: What leadership characteristics contribute to the positive outcome of software development programs? The supplemental research questions focused on software practices. Research question 2: What software practices contribute to the development of successful projects? Research question 3: What software practices result in negative impacts to projects? These research questions guided and focused the study on the investigation and identification of processes and practices that lead to successful software development programs and the impacts of leadership approaches.

Woolridge et al. (2009) observed proper leadership planning and management may decrease the project failure rate by providing effective scope definition approaches. Subject matter experts in the field of software engineering have investigated the effectiveness of software processes (Boehm, 2006; Kenett & Baker, 2010). Additionally, scholars have researched improvements in processes for software development (Beadell, 2009; Bonner, 2008; Sun, 2008) or improvement in leadership capabilities (Early, 2006; Jain, 2007; Johnson, 2008). The research questions for this study focused on effective integration of leadership capabilities and software processes for success. A grounded theory study focusing on leadership and process added to the existing body of knowledge by integrating the leadership and process dimensions for software engineering.

Theoretical Framework

This research study was under the broad theoretical areas of organizational paradigms, leadership, productivity, and process improvement. Effective leadership is critical to the continued success and growth of any organization (Miller & Desmarais, 2007). In the field of IT, leadership must modify and improve the leadership processes to meet the challenges of ongoing growth and complexity in the technical environment (El Emam & Koru, 2008; Mukherjee, 2008).

Leading a diverse and distributed team of software engineering experts in developing and implementing successful projects requires the effective balancing of resources, environment, and processes (Cusumano, 2008). The open system paradigm provided a framework for integrating the leadership approaches to resources, environment, and processes (National Defense University, 2009). This research study investigated software development leadership in the framework of open systems.

The main research question 1 investigated the leadership environment for software development to identify characteristics that support successful programs. The secondary research question 2 and research question 3 investigated the processes and resources applied to development programs. The open system paradigm combines environment, processes, and resources to develop a complete system approach (Bloch, 2008; Scott & Davis, 2007). This methodology provided a framework for investigating software engineering initiatives to develop theories and methods for leadership success.

The field of software engineering and IT contain many research studies investigating methods for improvement in program initiatives (Early, 2006; El Emam & Koru, 2008; Gefen et al., 2006; Jain, 2007). Research studies have been conducted to explore process improvement (Boynton, 2007; Jain), development methodologies (Agerfalk & Fitzgerald, 2006; Bonner, 2008; Cantor, 2002), and leadership approaches (Boseman, 2008; Denning, 2007; Kerzner, 2009). This research study added to the software engineering body of knowledge for leadership approaches for successful software development programs. The open system paradigm provided a framework for integrating the process improvement initiatives, development methodologies, and leadership approaches.

The software engineering community consists of numerous paradigms, processes, and approaches (Adams, 2008; Boehm, 2006; Kenett & Baker, 2010). Numerous perspectives exist on the most effective process, method, or approach (Boehm; Hinchey et al., 2008; Kenett & Baker). Some experts support traditional methodologies and structured processes (Adams; Cantor, 2002; Hinchey et al.) whereas other experts support less structured approaches and more agile processes (Agerfalk & Fitzgerald, 2006; Bonner, 2008; Johnson, 2008). Although these studies address individual aspects of success for software development, a gap exists on the integration of leadership approaches to individual components of the numerous methodologies and paradigms. Achieving success in software development often depends on the ability of leadership to select and effectively apply selected methodologies and approaches (Adams; Basili et al., 2008).

In the software engineering organization, success depends on the identification and implementation of successful leadership theories and paradigms (Jianguo, Jinghui, & Hongbo, 2008). As observed by Warzynski (2005), organizational and leadership "development is a process in which people, work processes, structures, and technologies are developed, integrated, and aligned to strengthen an organization's economic performance or increase its capacity to adapt and respond effectively to the environment in which it operates" (p. 338). Developing a software engineering leadership approach that encompasses the open system paradigm theories provides a framework for continual evaluation, assessment, and improvement (National Defense University, 2009).

The open system paradigm integrates resources, environment, and processes to develop a cohesive synergistic approach and structure to leadership (National Defense University, 2009). Open system approaches focus on a larger context above the immediate benefits to the organization by incorporating the impacts to individuals, organizations, and society (Scott & Davis, 2007). Leadership following the open system paradigm integrates the critical success attributes of self-regulation, knowledge sharing, environmental awareness, flexibility, diversity, innovation, and creativity (National Defense University).

To maintain pace with the expanding and changing technical environment for software engineering, leadership must embrace the tenets of self-regulation and self-maintenance (Jianguo et al., 2008). The open system paradigm emphasizes the importance of monitoring, modifying, and correcting processes throughout the program life cycle (Bloch, 2008). The open system paradigm focuses on increasing production and productivity and encompasses the concept of continual process improvement by identifying opportunities for improvement, actively pursuing improvements, and changing paradigms as required (Bloch).

Successful and innovative leadership for software development requires effective internal and external communication (Bharadwaj & Saxena, 2006). In the open system paradigm, enhanced communication is a result of the emphasis on information sharing and knowledge development (Scott & Davis, 2007). This environment promotes information flow between internal environments and external entities as required (Landaeta, 2008). Leadership encourages open, flexible, and adaptive information flow and communication throughout the structure and environment to foster improved development efforts (Stephenson & Sage, 2007). As noted by Johnson-Cramer, Parise, and Cross (2007), developing networked and boundless information flow approaches promoted effective interagency collaboration, learning, and knowledge sharing.

Implementing leadership processes for communication within the open system paradigm improves the effectiveness of the organization by disseminating the vision, sharing feedback, involving employees, integrating stakeholders, and improving innovation (Lewis, 2006). Open system leadership is flexible, adaptable, willing to take risks, and celebrates diversity (Claiborne, 2007). Flexible organizations seek opportunities for improvement, are willing to adapt when opportunities arise, and accept risks for growth and improvement (Claiborne).

Leadership in the open system paradigm focuses on developing an effective integration of resources, environment, and processes to meet defined goals and objectives (Lewis, 2006). Innovation, creativity, and information sharing flourish through leadership integration of effective processes for collaboration (National Defense University, 2009). Open systems leadership implements and evolves an environment which encourages, fosters, and implements initiatives to introduce new processes, improve existing processes, and remove ineffective processes (Claiborne, 2007). Through the framework of the open system paradigm, leadership for software development can implement integrated approaches for success, embrace opportunities for improvement, remain competitive in the technical environment, and enhance the success rate for software engineering initiatives (Claiborne; Lewis; Stephenson & Sage, 2007).

Definition of Terms

This research study presented terms and concepts which might have unique definitions related to leadership for software engineering. The following terms, in alphabetical order, focus on software development leadership.

Agile software development. The "practice of software development that assumes short development iterations with a fast realization of an executable software system that contains only a small part of functionality according to direct user input" (Pozgaj, Sertic, & Boban, 2007, p. 75).

Basic software process. A basic process for software engineering is a "basis for the development of computer programs with the use of a body of knowledge, LC standards, the infrastructure, and management in a developer organization" (Lavrishcheva, 2008, p. 331).

Canceled project. A canceled project is a development effort that did not deliver any usable functionality or a project terminated before completion (El Emam & Koru, 2008).

Challenged project. "Challenged projects are completed and approved projects that are over budget, late, and with fewer features and functions than initially specified" (Dalcher & Benediktsson, 2006, p. 51).

Capability maturity model (CMM). "A system for measuring the quality of the processes used within a software development organization. The CMM provides a means for the qualitative evaluation of processes without the need to follow a specific methodology" (Douglas, 2006, p. 28).

Collaboration. Collaboration involves the focused and coordinated effort of a collection of individuals with a shared understanding and common objective (Hadar, Sherman, & Hazzan, 2008).

Collaborative software development. Collaborative software development encompasses "the multiple teams, working for multiple organizational units within the same or different companies" (Mohtashami, Marlowe, Kirova, & Deek, 2006, p. 20).

Culture. Several definitions exist for culture. In an organizational context, "culture refers to the entire organization, its values, strategic goals, and the formal and informal systems in place that guide managers and employees in everyday work life" (Lindbom, 2007, p. 101).

Defining the process. For software engineering, "defining the process means that all the activities to be performed have to be clearly stated, including the order in which they are to be performed and when they are considered complete" (McManus & Wood-Harper, 2007a, p. 316).

Failed project. A failed project is "canceled before completion, never implemented, or scrapped following installation" (Dalcher & Benediktsson, 2006, p. 51).

Key process area (KPA). One definition for KPA stated a KPA "contains the goals that must be reached in order to improve a software process. A KPA is said to be satisfied when procedures are in place to reach the corresponding goals" (McManus & Wood-Harper, 2007a, p. 323).

Management process. The term management process "refers to the activities that are undertaken in order to ensure that the software engineering processes are performed in a manner consistent with the organization's policies, goals, and standards" (Institute of Electrical and Electronics Engineers [IEEE], 2004, p. 8-2).

Project. Projects consist of multiple attributes and concepts. For software engineering a project is:

> any series of activities and tasks that: have a specific objective to be completed within certain specifications, have defined start and end dates, have funding limits (if applicable), consume human and nonhuman resources (i.e.., money, people, equipment), [and] are multifunctional (i.e., cut across several functional lines). (Kerzner, 2009, p. 2)

Project management. Project management is the management over the "development of a project, using a management theory adapted to the type of project and processes" (Lavrishcheva, 2008, p. 330).

Risk management. "Risk management is a routine practice of software development and project management. It deals with anticipating, preventing, and mitigating problems arising in the software product, project, or process, including difficulties in personnel, communication, and coordination" (Mohtashami et al., 2006, p. 20).

Software development effort. According to Agrawal and Chari (2007), software development effort is:

> The total effort beginning with the end of the requirements specification stage until the end of customer acceptance testing. It includes effort during high-level

> design, detailed design, coding, unit testing, integration testing, and customer acceptance testing. (p. 148)

Software development methodologies. Erdogmus and Williams (2003) observed "methodologies, or processes, are prescribed, documented collections of software practices (specific methods for software design, test, requirements documentation, maintenance, and other activities) required to develop or maintain software" (p. 284).

Software engineering. Several definitions exist for software engineering. A basic definition is "the application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (IEEE, 2004, p. 1-1). An expanded definition is:

> A system of methods and means of programming, engineering of planning and team processes, management of manufacturing computer software systems (software support, applications, families of systems, and software projects), methods of measurement and estimation of the compatibility of their various characteristics as to their conformity with customer's requests. (Lavrishcheva, 2008, p. 325)

Software engineering management. Software engineering management can be "defined as the application of management activities – planning, coordinating, measuring, monitoring, controlling, and reporting – to ensure that the development and maintenance of software is systematic, disciplined, and quantified" (IEEE, 2004, p. 8-1).

Software intensive system. An IT system where "software is a major component and that much of the functionality is achieved via software rather than hardware implementations" (Hinchey et al., 2008, p. 55).

Software process. “A set of activities, methods, practices, and transformations which people use to develop and maintain software and the associated products” (IEEE, 2004, p. 6-5).

Software project. “A unique and integrated product that represents collections of realized objectives, solved problems, and obtained results of activity that satisfy the requirements of the customer of the project” (Lavrishcheva, 2008, p. 327).

Assumptions

This research study incorporated the following assumptions. The first assumption was respondents to the interview questionnaire will provide honest responses to the open-ended questions on leadership and software engineering processes. Failure to provide honest responses may influence the study validity and results of the qualitative grounded theory study (Neuman, 2003; Shank, 2006).

The second assumption was respondents to the study have experience in leadership, process improvement, or software engineering. Respondents without knowledge in the area of study could result in inappropriate responses affecting the accuracy of the data review, analysis, and theory development (Bogdan & Biklen, 2007; Neuman, 2003; Shank, 2006). Respondents to the interview questionnaire had personal experience in software development, software methodologies, software processes, and leadership impacts. Skip logic was employed in the electronic interview questionnaire to remove and eliminate responses for participants not meeting the sampling criteria.

The third assumption was the respondent who returned the questionnaire was the individual who completed the questionnaire and the observations and perceptions are based on the personal experience of the participant. A potential existed for participants to have the questionnaire completed by other individuals. The data collected for analysis may not be accurate if an individual other than the respondent completes the interview questionnaire (Bogdan & Biklen, 2007; Shank, 2006).

Scope and Limitations

Several limitations could affect the validity of the research, data analysis, data collection, and study conclusions. The study scope was limited to research and development organizations in Alabama. The data analysis and study results may reflect individual perceptions and opinions based on the operations in the localized area, economic impacts, and job stability. Qualitative studies do not “entail the sampling procedures or sample size required to generalize systematically to some wider population or context” (Schram, 2006, p. 87). The study did not attempt to generalize the results to varying populations or geographic locations.

As Creswell (2008) observed “the intent of qualitative research is to establish the detailed meaning of information rather than to generalize the results and standardize the responses” (p. 141). Qualitative research focuses on investigating a depth of information on the research topic instead of simplifying the understanding of a phenomenon (Shank, 2006). In this setting, the findings are not generalized to the population but rather are applicable to a particular phenomenon (Bogdan & Biklen, 2007). The objective of this study was to identify trends and themes using grounded theory methodology that may be used by other software development organizations and the results may not be generalized to varying populations (Creswell).

A potential limitation for the study was the implementation of an Internet interview questionnaire to collect participant responses. The availability of participants may not yield a sufficient sample size to result in saturation or meaningful conclusions. The study invitation was provided to the organization population of 600 employees with a response rate of 12% or 71 respondents. In grounded theory research, small sample sizes are common and do not prevent the formulation of theories grounded in the raw data (Creswell, 2008; Suzuki, Ahluwalia, Arora, & Mattis, 2007).

Glaser and Strauss (1967) noted saturation results when the responses are not generating any additional findings. The electronic interview questionnaire provided data based on the response rate and time allocated for participation. Grounded theory research seeks “themes that will eventually serve as the bases for theory” (Shank, 2006, p. 150). Creswell (2008) and Suzuki et al. (2007) observed that for qualitative research small sample sizes often result in a rich data set for evaluation. The researcher did not seek to obtain saturation, but rather a rich data set for evaluation and analysis.

Respondents to the questionnaire may not have exercised due diligence in providing accurate representations and perceptions. Failure of the participant to represent best effort in responding to the questionnaire could influence the validity of the study results (Shank, 2006). The study results reflect the participant’s perceptions, views, and beliefs during the timeframe of interview questionnaire distribution (Creswell, 2008; Neuman, 2003).

The data collected was evaluated for common themes and trends for data analysis by the researcher. Shank (2006) observed “any thematic analysis will reach saturation” (p. 150). Due to the time constraints for the present study, a static sampling method was used in which all data was gathered prior to analysis (McCleaf, 2007; Polkinghorne, 2005). The researcher used the collected data for data analysis to develop theories on leadership for software development initiatives.

Delimitations

This qualitative research study involved research and development organizations in the state of Alabama. The study focused on obtaining leadership and software development team views and perceptions on software engineering techniques, software development process, and leadership approaches. This study attempted to address the leadership processes for improvement in software engineering projects.

To assure respondent confidentiality, descriptive information such as name, age, gender, race, specific job title, and organizational division was not collected. The questionnaire only collected basic demographics related to organizational role, years of software engineering experience, and years of leadership experience. The interview questionnaire did not provide the opportunity for the researcher to ask additional clarifying or probing questions to obtain expanded participant views (Creswell, 2008; Shank, 2006). Although the study results are limited to perceptions on software engineering leadership in a research organization in Alabama, the research could be repeated in other organizations and geographic areas.

Summary

Chapter 1 presented an overview of the impacts of failed and unreliable software development programs. Software exists in every aspect of life and has the potential for minor to catastrophic impacts (Sommerville, 2007). From common consumer products to complex flight navigation systems, software affects every individual, organization, and society (Schneidewind, 2007; Stackpole, 2008). Leadership for software engineering strives to reduce cost while improving software performance, reliability, and availability (Pressman, 2010; Schneidewind; Sommerville).

In 2009, the Standish Group reported project success rates of only 32%, a decrease from the 35% reported in 2006, with failure rates increasing from 19% in 2006 to 24% in 2009 (Rubinstein, 2007; The Standish Group, 2009). These results emphasized the need for evaluation and enhancement of existing software development leadership processes (The Standish Group). The implementation of leadership processes for software development may contribute to project success and improved product performance.

The qualitative grounded theory research study investigated leadership approaches to software engineering to identify processes for improvement to reduce project failures, cost overruns, and reliability issues in software development programs. Improved software engineering leadership approaches for life cycle development may result from the identification of process which foster successful project outcomes, process which result in negative impacts, and process which require modification. The research study focused on the theory of open system paradigm for leadership effectiveness to investigate the environment, resources, and processes for effective software engineering. Chapter 2 contains a review of the history of software engineering, software life cycle methodologies, software process improvement, distributed software development, leadership paradigms, and leadership capabilities.

CHAPTER 2: REVIEW OF THE LITERATURE

Chapter 2 includes a review of literature relevant to the research study focusing on leadership for software development programs. This qualitative grounded theory study investigated the leadership processes for successful software development programs. The data collected was reviewed to develop theory on leadership approaches and processes for successful development initiatives. The review of the literature included discussion of existing theory and methodology for leadership and software engineering.

The chapter includes an overview of software engineering history and reviews the topics of life cycle methodologies, process improvement, and distributed software development. The chapter also includes an overview of leadership paradigms and capabilities focusing on key roles, characteristics, and concepts contributing to effective leadership. The leadership theories of transactional and transformational paradigms are presented noting the previous research identifying key components and characteristics of each paradigm. The purpose of this study was to investigate leadership approaches and software development methodologies to determine processes which are successful, processes which are negative impacts to performance, and processes which require update to enhance program development initiatives.

Title Searches, Articles, Research Documents, and Journals

The literature review contains information from peer-reviewed journal articles, scholarly books, theses, dissertations, professional websites, and governmental websites. Searches were conducted for information relevant to the research topic using the online databases EBSCOhost, ProQuest, Gale Power Search, ABI/INFORM, ACM Digital Library, Best Practices Benchmarking Reports Repository, Business Insights, Emerald, SAGE, Faulkner's Advisory on Computers and Communications Technologies, and IEEE Computer Society Digital Library. Database keyword searches included software engineering, distributed teams, distributed software development, software development, leadership development, leadership methodology, software methodology, leadership processes, transactional leadership, transformational leadership, process improvement, software life cycle, and software risk.

The database searches resulted in abundant literary sources related to software engineering leadership as well as additional topics and sources to investigate. The review of scholarly books, theses, dissertations, professional websites, and governmental websites provided background information on software engineering, process improvement, and additional sources for investigation. Local public libraries and databases were used to investigate additional sources and obtain scholarly books for review. The database and library searches produced hundreds of documents for review and evaluation. Documents were reviewed for applicability and relevance to the research study on software engineering leadership.

History of Software Engineering

In the 1950s, software development focused on developing programs for large mainframe systems (Boehm, 2006). Computer equipment and processing time were expensive, requiring programmer review and desk check of coding algorithms prior to use on the computer system (Boehm). In this era, software development concentrated on coding machine instructions for the computer to interpret to obtain desired results (Boehm). Specialized individuals with backgrounds in physics or mathematics developed computer programs but did not apply engineering principles to code development (Boehm; Yang & Mei, 2006).

Organizations began to realize it was easier to change software than hardware and programming became a process of code and fix until the desired solution set was realized (Boehm, 2006). As programs became more complex, the process of code and fix resulted in reduced quality and reliability and often increased cost (Boehm). This approach led to the software crisis of the 1960s and identified the need for a disciplined approach to software programming (Pressman, 2010; Sommerville, 2007).

Prior to the 1960s, the term programming represented the process of developing computer or machine code for use in computer systems (Wirth, 2008). As the complexity of computer components increased, the complexity of programming increased resulting in the emergence of concepts for a structured approach to programming (Pressman, 2010; Sommerville, 2007). The term software engineering was introduced after a North Atlantic Treaty Organization (NATO) conference, “referring to the highly disciplined, systematic approach to software development and maintenance” (Wirth, p. 32). The conference highlighted the problems of designing and developing complex computer systems (Wirth). The emergence of software engineering was a result of the realization that better methodologies, processes, and tools were required to develop software and combat the growing software crisis (Sommerville; Wirth).

In the 1970s, the emphasis for software engineering was on developing structured approaches and languages for completing complex software tasks (Boehm, 2006). Software engineering began to focus on improving techniques through formalized methods which emphasized requirement development and design analysis before initiating software coding (Boehm). The concept of structured programming began to gain acceptance and led to the development of operating systems and tools to support the structured software engineering programming environment (Wirth, 2008).

With the introduction of microcomputers in 1975, computers became affordable for business and home use (Boehm, 2006). Before this time, the majority of computers were mainframe computers used in large organizations and universities (Boehm). The microcomputer provided resources available to all individuals (Yang & Mei, 2006).

With this growth in the computer market, the demand for efficient software programs increased (Yang & Mei, 2006). The field of software engineering became critical to the development of applications across multiple disciplines and levels of complexity (Mahoney, 2008). As the requirements increased so did the complexity of software systems and the need for processes to assist in the development life cycle continued to grow (Mahoney).

The 1980s continued to refine the best practices and software engineering began to focus on improvement in productivity and scalability (Boehm, 2006). The demand for processes, tools, and techniques to support the disciple of software engineering continued to evolve (Boehm). Structured methods and processes began to emerge as the catalyst for improving productivity (Boehm). Software processes began to emerge focusing on developing modular components for reuse in other applications to reduce cost and increase productivity (Boehm). The introduction of structured life cycle models and programming languages gave rise to a new paradigm centered on structured methods and processes (Mahoney, 2008).

In the 1990s structured methodologies, life cycle models, and process models began to emerge as the standards for software development (Boehm, 2006; Ebert, 2008; Sen & Zheng, 2007). The growth in computer resources resulted in a demand for readily available software solutions to match the growth in computing power (Boehm). Software became a critical component of company success with increasing demands for development and delivery of solutions to the consumer (Boehm).

The field of software engineering became a discipline characterized by increasing demands, work performed under time pressure, and reduction in code quality (Wirth, 2008). The resulting software represented inefficient code resulting in reduced quality and performance (Boehm, 2006). The need for processes throughout the software life cycle emerged as the solution to combat reduction in quality, reliability, and maintainability issues (Boehm; Ebert, 2008). In this framework, the software industry began to focus on engineering software through processes instead of developing software to meet market demands (Ebert; Wirth).

Since 2000, the field of software engineering has realized the need to foster continual improvement through the application of best practices and processes for software development (Boehm, 2006). The integration of people, tools, and processes fosters increased technology introduction (Ebert, 2008). In this framework, software engineering remains focused on continual process improvement as the standard for development programs (Ebert). Software engineering continues to evolve as a field focused on developing quality products using software life cycle models, techniques, and processes (Pressman, 2010; Sommerville, 2007).

The application of processes continues to be a critical component for the field of software engineering (Sommerville, 2007). Although the availability of processes, methodologies, and life cycle models for software development continues to increase, the discipline of software engineering has not realized a substantial reduction in program failure rates (Cerpa & Verner, 2009; Dalcher & Benediktsson, 2006; Gottesdiener, 2008; Horn, 2009; Mizell & Malone, 2007; Mukherjee, 2008; Pino et al., 2008; Sommerville; The Standish Group, 2009; Xu & Brinkkemper, 2007). The Standish Group has been conducting research on software project success rates since 1994 (Johnson, Boucher, Connors, & Robinson, 2001). Although some improvements were realized through 2006, the 2009 research revealed failure rates are again continuing to rise emphasizing the need for identification of improved processes in software engineering (The Standish Group).

Software Life Cycle Methodologies

The complex and evolving nature of software systems (Elfatatry, 2007; Hadar & Leron, 2008) requires software engineering methodologies remain agile and adaptable to meet requirements (Sommerville, 2007). Whereas software engineering incorporates a systematic and disciplined approach to software development, the processes must be adaptable to meet the diverse set of applications (Pressman, 2010). Software life cycle methodologies provide a framework for software practices, techniques, actions, and management in an integrated model (Harris, Aebischer, & Klaus, 2007; Peslak, Subramanian, & Clayton, 2008; Pressman).

Software methodologies provide a basis for each phase of software engineering while allowing flexibility in the application and flow of processes within the structure (Harris et al., 2007; Peslak et al., 2008; Pressman, 2010). “The skill set focusing on the life cycle of software engineering projects is critical to both understanding and practising *[sic]* sound development and management" (Benediktsson, Dalcher, & Thorbergsson, 2006, p. 87). The software life cycle model provides the engineering methodology for converting requirements into implemented software applications (Sen & Zheng, 2007). Methods and approaches to the software engineering life cycle include waterfall development, model driven development, and agile development (Elfatatry, 2007; Peslak et al.; Pressman).

Waterfall Development

The waterfall model was introduced in the 1970s as a methodology for performing software development to meet government contracting requirements (Harris et al., 2007; Larman & Basili, 2003). The model provided a sequence of phases for software development focusing on design, development, and requirement analysis (Harris et al.; Larman & Basili). The waterfall model provided a framework for conducting software engineering activities in development stages (Harris et al.; Sen & Zheng, 2007). Each stage of the model focused on an individual piece of the development life cycle such as design, code, implementation, or testing (Harris et al.; Sen & Zheng). As each stage is completed the focus shifts to the next phase of development (Harris et al.; Sen & Zheng).

In the waterfall method, the approach to software development became structured and centered on completing individual milestones prior to full project completion (Harris et al., 2007; Sommerville, 2007). This model was originally interpreted to represent a strict sequence of phases in a specific order for the development life cycle (Harris et al.; Sommerville). As processes began to evolve, the waterfall model became the basis for other variations of the phased development approach to software (Guntamukkala, Wen, & Tarn, 2006; Rajlich, 2006).

Research revealed improvements could be realized in the methodology if the phases overlapped for implementation in smaller increments as opposed to completing an entire phase for each program (Guntamukkala et al., 2006; Rajlich, 2006). This realization resulted in modifications to the waterfall paradigm (Aken, 2008; Pressman, 2010). Variations of the waterfall model include the V system model, incremental models, evolutionary models, and spiral models (Aken; Pressman). Each of these models provided a framework for focusing on phased development during the life cycle for software development (Aken; Pressman).

Evolutionary models provided a life cycle methodology fostering development of software in stages with the goal of providing early products for assessment and evaluation (Boehm, 1988; Pressman, 2010). Products are incrementally evaluated and evolved to meet end user requirements (Boehm; Pressman). In this approach, an increment of the operational product is developed and provided for evaluation (Boehm; Pressman). The feedback obtained is used to modify and improve the product by repeating the development stages to evolve the product to the final implementation (Boehm; Pressman). The objective of evolutionary models is to develop quality software using flexible and iterative approaches (Boehm; Pressman). Incremental and spiral models are two specific instantiations of an evolutionary model (Boehm; Pressman).

In iterative or incremental development, the project is broken down into a series of activities each represented by the waterfall model (Siddiqui, Hussain, & Hussain, 2006). This approach allows the software development team to perform successive refinements during the life cycle to develop and deliver the product (Siddiqui et al.). The "iterative approach enables the customer to evaluate the software increment regularly, provide necessary feedback to the software team, and influence the process adaptations that are made to accommodate the feedback" (Pressman, 2010, p. 69). Incremental development models provide benefits to program leadership over traditional waterfall approaches including early problem resolution, reduced rework, improved reliability, early return on investment, and improved user satisfaction (Benediktsson et al., 2006).

The spiral software development model follows the phased approach of the waterfall model with an additional emphasis on risk analysis and mitigation (Boehm, 1988; Hashmi & Baik, 2007). In the spiral model, individual increments of functionality are implemented following the defined stages of development (Siddiqui et al., 2006). The model continues to refine increments, adding functionality until the development of the product is complete (Siddiqui et al.). Each spiral iteration results in more functionality for the software system (Siddiqui et al.).

The risk component of the spiral model focuses on evaluating the impacts at each increment of the spiral to revise completion estimates and to determine if the project should proceed (Aken, 2008; Rajlich, 2006). Early research on the implementation of the spiral model reported software team productivity increases of at least 50% over the traditional waterfall model (Boehm, 1988). Spiral model development approaches increased the focus on software risk management and the integration of risk planning into the software phases (Hashmi & Baik, 2007).

"The Spiral Model proposes a cyclic approach for incrementally growing a system's degree of definition, design, and implementation while decreasing its degree of risk" (Chatterjee, 2008, p. 613). The integration of risk planning into the life cycle model provided an additional component for enhancing the quality and efficiency of development efforts (Boehm, 1988). The variations of the waterfall model provided a phased approach to development through defined finite stages of activities (Guntamukkala et al., 2006).

Model Driven Development

Software development programs are often viewed as wasteful activities (El Emam & Koru, 2008; Nevo & Wade, 2007; Xu & Brinkkemper, 2007). Projects are canceled, developed code is not used, and applications are not portable to other systems (El Emam & Koru; Nevo & Wade; Xu & Brinkkemper). A proposed solution to these software issues is to embrace a development paradigm incorporating a level of abstraction above the code level to enhance the flexibility and reusability of developed applications (Edwards, 2003). This level of abstraction is achieved through the implementation of a model driven life cycle for software development (Benediktsson et al., 2006). In model driven development, models define and document the system product at each stage of system development (Pressman, 2010).

Models emphasize developing individual components for integration into a final system (Pressman, 2010). This development of visual representations of the system supports integration activities required for development of large distributed systems (Balasubramanian, Gokhale, Lin, Ahang, & Gray, 2006). "The model driven approach to system development facilitates better understanding of system requirements capture, design, construction, and generation" (Gorry, 2008, p. 87). Model driven development was an appropriate method of development for many information system applications because the abstraction and refinement of model layers supported the increasing size and complexity of distributed software programs (Guntamukkala et al., 2006). The model driven approach fostered a paradigm in which the models are as important as the artifacts developed, the problem and solution domain are developed at different levels of abstraction, and the levels are linked by defining relationships between models (Alam, Hafner, & Breu, 2008).

In 2001, the Object Management Group introduced a software development approached based on the principles of domain engineering for model driven development of software systems (Haustein & Pleumann, 2005). Model Driven Architecture (MDA) provided a set of guidelines for modeling system specifications, developing software code from modeling diagrams, and implementing models throughout the software life cycle (Edwards, 2003). This methodology implemented platform models independent of the specific technologies or platforms for implementation (Haustein & Pleumann). Platform specific models are derived and refined from the independent model until the system is appropriately defined before code implementation (Edwards).

Throughout the life cycle, the implemented MDA models support the software planning, development, enhancement, and maintenance of the system (Edwards, 2003). The MDA approach allows the software engineering team to focus on individual points in the development process by separating the details from the implementation (Gorry, 2008). MDA provides a model driven life cycle approach which is independent of the hardware application, supports flexibility, enables prototyping, and encourages reuse (Edwards).

In the 1990s, the collaborative work of object oriented analysis and design methodology experts produced the Unified Modeling Language (UML) (Pressman, 2010). UML combined aspects of each of the object oriented and structured design methodologies (Pressman). UML provided semi-formal graphical diagrams for software engineering process with a means for defining and modeling object oriented systems (Aoumeur, 2008; Pressman).

UML has been considered by some software engineers to be the standard modeling language for object oriented information systems analysis and design (Jakimi & Elkoutbi, 2009; Rasulzadeh, 2008). “A major segment of the software engineering community has adopted the Unified Modeling Language (UML) as the preferred method for representing analysis and design models” (Pressman, 2010, p. 89). Some researchers believe UML does not incorporate formal methods critical to improved and automated software engineering processes (Benediktsson et al., 2006; Rasulzadeh). The choice of which approach to use is dependent upon the application and organizational strategy (Guntamukkala et al., 2006; Pozgaj et al., 2007).

Agile Development

Final software products are successful if they provide the required functionality, meet delivery schedules, and maintain budget requirements (Nasution & Weistroffer, 2009). The traditional waterfall based development approaches have been viewed as contributing to the perceived failure of software programs due to the structured, rigid, and segmented phases (Benediktsson et al., 2006; Rajlich, 2006). Projects following these life cycle models are often viewed as exceeding cost, exceeding schedule, and failing to deliver required functionality (Nasution & Weistroffer).

Agile development methods focused on improving performance through providing methods for quick response to changing environments and requirements (Aken, 2008). Agile development methods focus on short development iterations to produce executable software with partial functionality (Pozgaj et al., 2007). The product end user evaluates the partial product and provides feedback for use in the next iteration (Clutterbuck, Rowlands, & Seamons, 2009). User requirements are refined and updated to deliver a product that provides the desired functionality and meets user requirements (Clutterbuck et al.).

According to Keston (2008), agile development provides software engineers with flexibility in work execution approach. Agile moves away from traditional structured methodologies to embrace an approach focused on product evaluation instead of measuring documentation development or milestones achieved (Clutterbuck et al., 2009). In the agile philosophy, measured and observed iteration product quality defines program progress (Keston). In this approach, the software engineering team refines requirements and defines system functionality in increments involving stakeholders at each phase (Calabrese, 2008; Clutterbuck et al.). This methodology fosters flexibility and adaptability in the software life cycle approach (Calabrese; Clutterbuck et al.).

The agile development approach does not focus on one set of practices or processes but rather on the shared principles for the development life cycle (Clutterbuck et al., 2009). The key principles in the agile development methodology are iterative development, small steps, customer involvement, communication emphasis, small teams, pragmatic development, and testing (Keston, 2008). Agile life cycle processes embrace these philosophies to streamline the process over traditional formal methodologies (Aken, 2008).

An advantage of the agile method over the traditional waterfall model is the frequent delivery of executable software systems for evaluation (Clutterbuck et al., 2009; Pozgaj et al., 2007). The agile method for software development supports the evolving and dynamic technical environment associated with software development for information systems (Nasution & Weistroffer, 2009). Research conducted by Kendall et al. (2008) on scientific software development applications noted agile methods were more effective than traditional formalized approaches.

Agile methods provided greater flexibility, improved user input, and increased product quality (Calabrese, 2008). Some software engineering practitioners believe agile methods are difficult to apply to large projects consisting of diverse teams with fixed costs and schedules (Douglas, 2006; Guntamukkala et al., 2006). These programs tend to align with the traditional methodologies and approaches to program development (Douglas; Guntamukkala et al.).

Although numerous life cycle methodologies exist, no method provides a solution applicable to every application (Benediktsson et al., 2006; Eldai, Hassan, Ali, & Raviraja, 2008; Guntamukkala et al., 2006). The appropriate method depends on many factors including the type of development, team structure, project complexity, and performance parameters (Benediktsson et al.; Eldai et al.; Guntamukkala et al.). “Most complex systems involve a mix of technologies. We have a wealth of competing architectural platforms. Experienced designers must make trade-offs and select from a variety of appropriate technologies for solving the task at hand” (Wirfs-Brock, 2008, p. 30).

Model driven methodologies are considered effective for large, integrated IT systems but are often considered inappropriate for real-time system applications (Gorry, 2008). Although model driven development provides a method for refining requirements and design independent of the target platforms, no specific approach to defining transitions between model layers exists (Edwards, 2003). Gorry noted the lack of transition definition as a drawback for real-time, embedded, safety-critical systems. In this type of development the hardware, interfaces, and platform must be an integral component of the software development process (Edwards; Gorry). MDA provides abstractions above the hardware and interface level needed for real-time, safety critical application development (Gorry).

Selecting and implementing a life cycle methodology is only part of the activity for software engineering (Cusumano, 2008; Erdogmus, 2008). Processes must be implemented at each stage of the life cycle to support program completion (Cusumano; Erdogmus). Process improvement approaches are key components of software development program success and are essential to software engineering life cycle and development methodologies (Agrawal & Chari, 2007; Miller, 2006).

Software Process Improvement

According to Boehm and Valerdi (2008), the software field is changing and refining through the introduction of new methods, components, languages, models, and concepts. Software engineering programs strive to increase productivity and quality while providing the software applications required by a growing technical society (Pressman, 2010; Sommerville, 2007). “Over the last two decades, the software engineering community has expressed special interest in software process improvement (SPI) in an effort to increase software product quality, as well as the productivity of software development" (Pino et al., 2008, p. 237). As observed by Stephenson and Sage (2007), remaining competitive and current in the technical marketplace requires continual process improvement.

Glass (2008) noted the difficulty in determining the current state of software engineering practice due to the diversity in software engineering. The software field consists of numerous system sizes, types, and objectives (Glass). In a field of such diversity, identifying one best practice, methodology, or process is not feasible (Benediktsson et al., 2006; Subramanian, Klein, Jiang, & Chan, 2009). The very nature of software engineering requires continual investigation and improvement to maintain pace with the state of software practice (Glass).

"Our biggest challenges are to figure out how to selectively prune the parts of the software engineering experience base that become less relevant, and to conserve and build on the parts with lasting value for the future" (Boehm & Valerdi, 2008, p. 80). To meet this challenge, software engineering organizations implemented process improvement initiatives (Agrawal & Chari, 2007; Miller, 2006). The most popular of these initiatives included the Carnegie Mellon University maturity models (Carnegie Mellon University Software Engineering Institute [CMU/SEI], 2006), International Organization for Standardization (ISO) initiatives (McManus & Wood-Harper, 2007a), and Six-Sigma (Boynton, 2007) approaches.

Capability Maturity Model (CMM)

The CMM provided an approach for understanding and analyzing the capability maturity of applied processes within an organization (CMU/SEI, 2006). CMM was developed with the goal of helping organizations improve process initiatives (Jianguo et al., 2008). CMM assisted organizations in determining current maturity level, identifying issues critical to process improvement, defining the software process, and implementing a software process improvement program (Galin & Avrahami, 2006).

The CMM provided “five evolutionary levels: (1) initial, (2) repeatable, (3) defined, (4) managed, and (5) optimizing” (Ramasubbu, Mithas, Krishan, & Kemerer, 2008, p. 439). As organizations move up the maturity levels, the effectiveness and control of the software processes are improved (McManus & Wood-Harper, 2007b). The maturity levels for CMM provide a framework and methodology for continual process improvement (McManus & Wood-Harper, 2007a). The goals at each maturity level provide the method for establishing maturity and increasing process capability within the organization (McManus & Wood-Harper, 2007a).

The CMM provided a framework for assessing the maturity of the development stages of the organization for definition, implementation, measurement, control, and improvement (Jianguo et al., 2008). The implementation of CMM focused on continual process improvement for the stages of software development (McManus & Wood-Harper, 2007b). As organizations move through the model levels, processes applied during the life cycle were further refined (McManus & Wood-Harper).

Along with the refinement of processes, the model emphasized continual process improvement (Jianguo et al., 2008). Progressing through the maturity levels to level five did not indicate the final processes had been achieved (Jianguo et al.; McManus & Wood-Harper, 2007b). Organizations improved efficiency and productivity through continual process improvement within the CMM framework (Jianguo et al.; McManus & Wood-Harper).

Research studies have revealed when organizations achieve high maturity levels, effort is reduced, quality is increased, and productivity is enhanced (Agrawal & Chari, 2007). Research conducted by Galin and Avrahami (2006) indicated CMM programs consistently improved performance in the software development metric areas of productivity, error density, error detection effectiveness, cycle time for completion, percentage of rework required, schedule fidelity, and return on investment. In conducting research on software review quality, Mishra and Mishra (2008) found implementing CMM processes resulted in a decrease in review time and improved resource utilization.

The SEI continues to improve the guidelines and framework for software engineering (Jianguo et al., 2008). The success of CMM for process improvement led to the development of additional models for other system areas (CMU/SEI, 2006). The Capability Maturity Model Integration (CMMI) superseded the CMM for system process improvement (CMU/SEI). CMMI provides a framework for total system engineering and the CMM is applicable in the general theoretical domain for software engineering (CMU/SEI).

Capability Maturity Model Integrated (CMMI)

The CMM success led to the development of models for other disciplines such as software acquisition, system engineering, and integrated product development (Kneuper, 2009). As these models were applied within organizations, problems with implementing and managing different models became apparent (Kneuper). Organizations desired one process improvement model for use in multiple focus areas (CMU/SEI, 2006). The

CMMI was developed by the SEI to address the problem of implementing multiple independent models (CMU/SEI).

CMMI combined various independent models for software, systems engineering, and integrated product development into a single improvement framework (Kneuper, 2009). "The combination of these models into a single improvement framework was intended for use by organizations in their pursuit of enterprise-wide process improvement" (CMU/SEI, 2006, p. 6). The CMMI provided guidelines for process improvement to define the artifacts and activities required across multiple organizational disciplines (CMU/SEI; Kneuper).

Although CMM and CMMI provided a framework for process improvement, some software engineering experts believe the models are overly bureaucratic and concentrate on the development of processes without considering factors such as budget, tools, and personnel who also contribute to program success (Probert et al., 2007). Process improvement initiatives are not limited to CMM and CMMI approaches (Persse, 2006). Organizations not embracing the CMM and CMMI methodologies can implement standards from the ISO or Six-Sigma initiatives for process improvement (Persse).

International Organization for Standardization (ISO)

The ISO developed quality assurance standards applicable to any business enterprise (Persse, 2006). The ISO 9000 standards provided a framework for business processes focusing on quality product development (McManus & Wood-Harper, 2007a). The ISO 9001 series for software engineering are "detailed standards, which covers design, development, production, installation, and servicing" (McManus & Wood-Harper, p. 320). Applications with a significant design aspect, such as software engineering applications, implement the ISO 9001 framework for process improvement (Persse). Companies implementing the ISO framework have completed an accreditation process verifying the standards are understood, documented, and implemented within the organization (McManus & Wood-Harper).

The ISO standards provided customers and end users with confidence that implemented processes for designing, developing, and managing software quality are effective (Persse, 2006). The software engineering industry has not fully embraced the ISO paradigm as an acceptable process improvement initiative for development (McManus & Wood-Harper, 2007a). The ISO 9000 series originated as a set of standards in the manufacturing industry, which embody generic process and practices (McManus & Wood-Harper). This generic focus and background in manufacturing resulted in many software engineering practitioners failure to embrace these quality standards for process improvement initiatives (McManus & Wood-Harper).

The International Electrotechnical Commission (IEC) and ISO jointly developed ISO/IEC 15505 for the assessment of technical application processes (Al-Qutaish & Al-Sarayreh, 2008; International Organization for Standardization [ISO], 2004b). This standard established an international standard in the area of maturity models (Al-Qutaish & Al-Sarayreh; ISO). The model framework focused on the software business areas of organization, management, engineering, acquisition supply, support, and operations (Al-Qutaish & Al-Sarayreh).

The ISO/IEC 15504 model defined six process capability levels of optimizing, predictable, established, managed, performed, and incomplete (International Organization for Standardization [ISO], 2004a). Technical organizations implement ISO/IEC 15504 to enable process improvement initiatives (Al-Qutaish & Al-Sarayreh, 2008; ISO). The ISO/IEC model provided the methodology for assessing an organizations capability to deliver products at each defined level of maturity (ISO).

Six-Sigma

Six-sigma is a business strategy implemented to improve the proficiency, profitability, quality, and efficiency of business processes (Persse, 2006). In six-sigma environments, the focus is on customers, processes, and employees to achieve improved levels of success (Persse). In the business environment, six-sigma techniques are used to develop and implement new applications to support improvement across the life cycle (Goztas, Baytekin, & Kamanlioglu, 2009). “Six sigma in essence is a management policy built upon data and facts where the main focus is at customer oriented operation, excellence and process management” (Goztas et al., p. 48). The six-sigma methodology provided a means for measuring quality and performance through process control (Boynton, 2007).

The six-sigma approach provided two methodologies to implement change (Boynton, 2007). The DMAIC model is implemented for existing process modification and the DMADV is implemented for new process or product initiatives (Boynton; Goztas et al., 2009). The DMAIC methodology focuses on defining what needs to be improved, measuring current approaches, analyzing current processes, developing an improvement plan, improving current processes, and repeating the process (Boynton). The DMADV model provides five phases with the final two phases emphasizing the introduction of new processes (Boynton). The five phases in this model define goals, measure customer requirements, analyze the process, design process improvements, and verify performance to meet customer requirements (Boynton).

The structured framework provided by the six-sigma methodology enables a method for effective process execution (Persse, 2006). Each of the phases focuses on individual aspects of a business process with the goal of improving efficiency, quality, and productivity (Keller, Marose, & Schussler, 2009). The concepts of six-sigma assist organizational leadership in developing a culture of continual improvement (Goztas et al., 2009). The six-sigma philosophy is applicable to business processes and the tenets of six-sigma are applicable to process improvement initiatives for the field of software engineering (Boynton, 2007).

Shenvi (2008) conducted research on applying the six-sigma methodology to software development process in an electronics organization. The organization researched concentrated on development of embedded software intensive systems for consumer use (Shenvi). Shenvi observed the satisfaction of the consumer was critical to continued organizational success. The organization studied applied the concepts of six-sigma to the software engineering processes for product design and development (Shenvi). The study indicated well planned applications of the six-sigma methodology result in improved requirement management, reduction in communication issues, reduced rework, and improved proficiency (Shenvi).

Each of the methodologies presented provides a unique focus and implementation approach (Persse, 2006). Each methodology provides a framework for process improvement initiatives in business environments (CMU/SEI, 2006; Goztas et al., 2009; ISO, 2004a; Persse). For software engineering, the continual challenges of designing and developing complex, embedded, software intensive systems requires the continual improvement and evaluation of processes for software engineering (Agrawal & Chari, 2007; Basili & Zelkowitz, 2007). The instantiation of a process improvement initiative coupled with an applied life cycle development methodology fosters an environment of increased performance, proficiency, reliability, quality, and successful software development efforts (Kneuper, 2009; Mishra & Mishra, 2008; Persse).

Distributed Software Development

Resources, information availability, and knowledge development are enhanced through the continued advances in computer technology, networking systems, and interface applications (Awazu et al., 2009; Neumann, 2008). Software development organizations implement activities through experts, data, information, and knowledge in the local and global environments (Neumann; Peppard, Ward, & Daniel, 2007). Distributed development activities provide the opportunity to enhance information sharing and team collaboration for improved product success (Bernstein & Haas, 2008). Distributed software development has emerged as a viable solution to the problems of "skill set availability, acquisitions, government restrictions, increased code size, cost and complexity" (Bird, Nagappan, Devanbu, Gall, & Murphy, 2009, p. 85).

The advances in technology, computer hardware, and networking capabilities require larger and more complex software system development efforts (Hadar & Leron, 2008; Kirova et al., 2008; Schneidewind, 2007). In this environment, many organizations perform software engineering activities implementing locally and globally distributed teams of experts collaborating throughout the development life cycle (Hadar et al., 2008). The growth in availability and functionality of computer applications and networking capabilities has resulted in increased demand for complex and innovative software systems to meet consumer demands (Hinchey et al., 2008). To remain current and competitive in the global marketplace, software engineering teams must work in a collaborative, distributed environment to deliver software applications to meet consumer quality, reliability, and functionality demands (Brown & McDermid, 2008; Lee, Delone, & Espinosa, 2006).

Communication and Collaboration in Distributed Teams

Hadar et al. (2008) defined collaboration as the focused and coordinated effort of a collection of individuals with a shared understanding to develop a successful problem solution. To support activities in the distributed, collaborative environment, software teams implement processes, methods, and practices to develop a common approach and philosophy for software development (Brown & McDermid, 2008). Research has been performed to investigate the challenges of software engineering in the distributed environment. Many of these studies emphasized the importance of developing and applying effective processes for team collaboration and software development (Bharadwaj & Saxena, 2006; Bird et al., 2009; Bose, 2008; Espinosa, Slaughter, Kraut, & Hebsleb, 2007; Kotlarsky, Oshri, & Willcocks, 2007).

Bharadwaj and Saxena (2006) conducted a study to identify the communication methods, tools, and processes for successful application in global software development teams. The organizations investigated implemented a distributed team structure of individuals across the organization and globe (Bharadwaj & Saxena). Although the distributed structure provided teams with increased flexibility and responsiveness, the communication and collaboration challenges were increased (Bharadwaj & Saxena).

The study results revealed the key concept in assuring program success in the distributed environment is to communicate knowledge and information to the management team, project team, and stakeholders (Bharadwaj & Saxena, 2006). Managing the communication processes for knowledge and information dissemination contributed to team and project success (Bharadwaj & Saxena). The effective management of communication and knowledge processes contributed to achieving the goals of the program (Bharadwaj & Saxena).

Bird et al. (2009) observed distributed software development incorporates additional challenges not presented in co-located teams such as inconsistent development environments, delayed feedback, lack of trust, and restricted communication. These factors not only affect team effectiveness, but also product quality (Bird et al.). The case study conducted by Bird et al. focused on evaluation of the processes implemented for distributed development teams and the influence on product quality.

The results of the Bird et al. (2009) study indicated product quality was not influenced by the distributed environment provided effective processes and practices for improved communication, coordination, and team cohesion were implemented. Effective practices identified included communication, consistent tool implementation, product ownership, common schedules, and organizational integration (Bird et al.). The study results demonstrated the importance of developing, identifying, and applying effective practices throughout the development life cycle (Bird et al.).

Kotlarsky et al. (2007) noted innovations in communication and networking have increased the use of distributed software development teams, creating challenges for leadership over co-located development initiatives. Distributed development projects are often large, complex activities requiring intensive communication and collaboration (Kotlarsky et al.). The success of these initiatives often depends on the technical and operational methodologies and processes implemented for communication and collaboration (Kotlarsky et al.). Achieving success in the distributed environment requires effective communication and collaboration (Bose, 2008; Espinosa et al., 2007). Communication and collaboration are all influenced by the social interactions developed by team members (Bird et al., 2009; Kotlarsky et al.; Lee et al., 2006).

The research by Kotlarsky et al. (2007) extended previous investigations on social interactions for distributed development by focusing on investigating the “processes through which social ties are created and renewed” (p. 10). The research results revealed the application of processes for social interaction should consider the current stage of the development team (Kotlarsky et al.). Different stages of development required different processes and techniques to achieve improved social interaction (Kotlarsky et al.). Although processes and techniques are available for implementation to increase team interaction, no one process or technique works in every development phase (Kotlarsky et al.). Distributed software team leadership must develop an integrated collection of processes for communication and collaboration which are selectively applied to meet the challenges of the specific team development phase (Kotlarsky et al.).

Formal and Informal Processes for Distributed Teams

“Large-scale software development requires a substantial amount of coordination because software work is carried out simultaneously by many individuals and teams, and then integrated into a single product” (Espinosa et al., 2007, p. 136). Development in this integrated environment presents challenges for synchronizing and coordinating activities

(Bird et al., 2009; Bose, 2008; Kotlarsky et al., 2007). The coordination challenges are further complicated when the development team is distributed across the organization or geographic locations (Bird et al.; Bose; Kotlarsky et al.). Problems and delays in software programs can result from improper management of established software development processes (Tesch et al., 2007). Determining the appropriate group of leadership and development processes is essential to success in the distributed software development environment (Espinosa et al.).

Espinosa et al. (2007) investigated the effectiveness of processes for distributed software development from the perspective of leadership and team members. The research study conducted interviews with team members and leadership in a distributed software development organization (Espinosa et al.). The greatest problem identified by study participants centered on coordination problems (Espinosa et al.). The coordination problems were broken into the categories of technical, temporal, and process (Espinosa et al.). For process coordination, 56% of the participants revealed problems in the effective management of software development processes (Espinosa et al.).

The Espinosa et al. (2007) study further revealed the specific process problems noted were based on individual perspectives. Only 41% of the technical staff noted process coordination problems whereas 100% of managers noted some type of process coordination issue (Espinosa et al.). The results revealed technical staff members were more concerned about coordination problems whereas managers were more concerned about managing software development processes (Espinosa et al.).

Software development teams in distributed environments can provide benefits such as reduced development costs, improved flexibility, increased productivity, and shared best practices (Agerfalk & Fitzgerald, 2006; Bose, 2008). Software development in the distributed environment provides additional challenges over collocated development that must be addressed through the application of processes and procedures (Cusumano, 2008; Monalisa et al., 2008). Distributed software development provides the advantages of multiple locations, increased talent base, increased quality, and lower cost (Bose). Through work distribution across different time zones, software development activities could continue around the clock (Bose; Cusumano). Managing a distributed team in this environment creates additional challenges resulting from the communication difficulties, work culture differences, and conflicts in organizational methodologies (Bose; Cusumano).

Bose (2008) conducted case study research to investigate the concepts in agile software development which could be effectively applied to software development in the distributed environment. The study investigated the solution strategies applied by 12 companies to overcome the challenges of distributed software development (Bose). In each case, the goal of the organization was to increase efficiency by improving repeatable processes (Bose).

The Bose (2008) study indicated a focus on processes for team selection, knowledge management, communication, and environment was essential to successful project outcomes. The most important observation noted “different solution strategies work for different companies based on available resources, intended outcome, and work culture” (Bose, p. 630). When focusing on improving distributed software development no specific process or practice works in every situation or life cycle phase (Bose).

Developing a comprehensive set of software engineering processes for selective implementation can support improved project performance (Bose).

Lee et al. (2006) conducted research of 22 global software development teams to analyze the key processes, inputs, and outputs implemented for program success. The main process investigated focused on communication and coordination among team members (Lee et al.). The results of the study by Lee et al. found “successful global software teams applied common principles in deploying coping strategies to enhance both flexibility and rigor in software development” (p. 38). The evaluation of the data collected by Lee et al. during interviews resulted in three general principles present in global software development.

The first principle noted in the Lee et al. (2006) study stressed the importance of defining standardized processes at the start of a program. Establishing processes early enables the team to make needed modifications more effectively and at a lower cost (Lee et al.). The second principle noted distributed software teams would implement and adapt the processes for specific tasks and will eliminate the ineffective processes (Lee et al.). The final principle observed teams in the global environment require the establishment of rigorous software development processes to meet the challenges of communication and coordination (Lee et al.). The established processes provide the framework for continued task execution without the need for coordination among all team members (Lee et al.).

Although agility and flexibility are trends in software development approaches, the research by Lee et al. (2006) demonstrated flexibility must be combined with discipline and rigor for process application to realize success in distributed software development programs. The study noted teams without disciple and rigor in the process for the software development life cycle could become inefficient and ad hoc in the global environment (Lee et al.). As noted in other studies (Bharadwaj & Saxena, 2006; Bird et al., 2009; Kotlarsky et al., 2007), the distributed team tailored processes and techniques to match specific applications and phases (Lee et al.).

Research on formal processes for distributed software development by Ramesh, Cao, Mohan, and Xu (2006) revealed a mix of formal rigid methods and flexible application resulted in successful programs. The dynamic business environment requires software development organizations design and implement software intensive systems more efficiently with a decreased time to market (Miller, 2006; Ramesh et al.). This environment requires flexibility in processes, applications, and methodologies (Ramesh et al.). Distributed development environments often achieve control through the development and implementation of formal processes (Ramesh et al.).

Ramesh et al. (2006) conducted research of three organizations to identify the effective application strategies for flexible software engineering processes. The research focused on investigating processes to align with the agile methodologies of continual analysis and improvement, knowledge sharing, improving communication, building trust, and verifying processes (Ramesh et al.). In each organization, leadership encouraged development processes that incorporated flexibility to support rapid development (Ramesh et al.). These processes also required periodic analysis and verification to assure the development process remained under control and disciplined (Ramesh et al.). The results revealed a careful balance of agile methodologies and distributed formal approaches foster improved communication, control, and trust across distributed teams (Ramesh et al.).

The studies in the literature on distributed software development provide evidence for developing formal disciplined processes and providing flexible, agile processes (Bharadwaj & Saxena, 2006; Bird et al., 2009; Bose, 2008; Espinosa et al., 2007; Kotlarsky et al., 2007). Throughout the literature on distributed software development, the common theme identified related to the concept of no single set of processes is applicable to every application, environment, and approach (Agerfalk & Fitzgerald, 2006; Bharadwaj & Saxena; Bird et al.; Ramesh et al., 2006). Achieving success in distributed software development requires the development of a collection of processes selected and tailored to match the specific application and development domain (Agerfalk & Fitzgerald; Bharadwaj & Saxena; Bird et al.; Ramesh et al.). As observed by Agerfalk and Fitzgerald in implementing processes for distributed software development, leadership must realize no single best method or process is available for incorporation. The important concept for leadership is the development of an integrated collection of processes and methods for implementation and tailoring to match the specific development effort, phase, project objective, and team composition (Agerfalk & Fitzgerald; Bharadwaj & Saxena; Bird et al.; Ramesh et al.).

Software Program Failure Rates

The Standish Group (2009) conducted independent research and analysis of IT project performance to evaluate the status and trends for software development programs. The study categorized projects as successful, failed, or challenged:

> Successful: The project is completed on time and on budget, with all features and functions originally specified. Challenged: The project is completed and operational, but over budget, late, and with fewer features and functions than

initially specified. Failed: The project is canceled before completion, or never implemented. (Johnson et al., 2001, p. 1)

The initial study, conducted in 1994, reported 16% of projects successful, 31% of projects failed, and 53% of projects challenged (Johnson et al.). The Standish Group repeated the study every 2 to 3 years with each report revealing gradual improvement through 2006 (Rubinstein, 2007).

In 2004, the Standish Group report revealed marked improvements over the 1994 results with 18% of projects successful, 29% of projects failed, and 53% of projects challenged (Boehm & Valerdi, 2008; Nevo & Wade, 2007). These improvements in program success rates, and declines in failure rates continued in the 2006 study (Boehm & Valerdi; Rubinstein, 2007). The 2006 Standish Group report revealed 35% of projects successful, 19% of projects failed, and 46% of projects challenged (Boehm & Valerdi; Rubinstein). These findings were an improvement over the initial 1994 study and the 2004 study results.

In 2009, the Standish Group study reported a reversal in the improvement trend with a marked decline in software development program success. The study reported 32% of projects successful, 24% of projects failed, and 44% of projects challenged (The Standish Group, 2009). These results revealed a significant decline in program success rates and an increase in program failure rates (The Standish Group). The failure rates were the highest reported in the last decade and indicated software development programs and leadership need to investigate new methods, procedures, and practices to improve the decline in successful software program rates and reverse the increase in failure rates (The Standish Group).

Leadership Paradigms

Software engineering is a complex, dynamic, and integrated activity which aligns business initiatives with development strategies (Adams, 2008; Pfleeger & Atlee, 2010; Sommerville, 2007) to produce software products characterized as "complex, changeable, and invisible" (Probert et al., 2007, p. 810). The management of software engineering teams in this environment requires effective leadership for organizational, product, and process objectives (Colbert, Kristof-Brown, Bradley, & Barrick, 2008; Desouza, Awazu, & Baloh, 2006). As observed by Erdogmus (2008), software development is an intellectual, human centered effort requiring effective leadership.

Leadership is not the direction of individuals for task completion but rather the effective application of leadership paradigms to motivate, support, inspire, and foster achievement of individual, project, and organizational objectives (Ilies, Judge, & Wagner, 2006; Northouse, 2010). Individuals demonstrate leadership when they are able to assess the capability of the available staff, provide employee support, provide direction when required, participate in team activities, and demonstrate enthusiasm for organizational objectives (Northouse; Taylor, 2007). Transactional and transformational leadership are two common paradigms often applied to software development initiatives.

Transactional Leadership

Transactional leadership focuses on building transactions or influential exchanges between leaders and followers (Boseman, 2008). In this leadership style, the leader provides rewards in exchange for effort, participation, productivity, and performance of followers (Boseman). Transactional leaders foster and develop effective relationship with employees to improve employee satisfaction and performance (Boerner, Eisenbeiss, &

Griesser, 2007). In this environment, employees observe the dynamic between benefits received and the quality of work performed (Boerner et al.). Transactional leaders define and reward performance based on individual tasks and expected productivity (Boerner et al.; Boseman).

Leaders in the transactional paradigm often provide rewards for positive performance and manage by exception (Boerner et al., 2007). In this environment, the leader and follower establish a strong bond and degree of trust (Boerner et al.). In this paradigm, employees will only perform to the level rewarded and will not provide additional effort (Boseman, 2008). In a transactional environment, individuals do not establish a common bond with leadership and often do not identify with organizational goals and objectives (Boerner et al.). The driving factor for individuals is to achieve expected performance to obtain rewards (Boerner et al.).

Transformational Leadership

Transformational leaders go beyond the reward for performance methodology of transactional leadership and attempt to engage all aspects of individual motivation (Denning, 2007). Transformational leaders convey to followers the importance of the individual to the organization by communicating the goals, objectives, and importance of each contribution (Boseman, 2008). Transformational leaders inspire, motivate, encourage, and generate enthusiasm for a common purpose (Denning; Northouse, 2010).

Organizations require skilled and effective leadership at all levels to ensure the understanding and achievement of goals and objectives (Northouse, 2010). The transformational leadership paradigm encourages creativity and innovation for individuals, teams, and the organization (Boerner et al., 2007). Transformational leaders exhibit “courage self confidence, passion and energy, as well as strong interpersonal skills, an ability to imagine different and better futures, an ability to communicate visions and a willingness to take risks” (Taylor, 2007, p. 28).

Transformational leaders integrate technical and personal characteristics and traits to encourage, innovate, and inspire individuals, teams, and organizations (Bass, 1999; Northouse, 2010). Transformational leadership focuses on obtaining the maximum potential from individuals through motivation, encouragement, and support (Tarabishy, Solomon, Fernald, & Sashkin, 2005). For individuals “superior performance is possible only through stimulating and motivating followers to higher levels of performance” (Masood, Dani, Burns, & Backhouse, 2006, p. 942). Transformational leaders motivate and encourage followers by exhibiting enthusiasm, communicating objectives, articulating a clear vision, openly communicating, defining expectations, and providing individualized support (Ilies et al., 2006; Masood et al.; Northouse).

For the transformational leader, innovation is critical to employee motivation, creativity, and discoveries (Northouse, 2010). Leadership must encourage innovation and support innovative processes and activities to foster creativity and performance improvement (Miller, 2006). According to Miller, businesses must embrace innovation and innovative leadership to survive in the competitive, global economy. Transformational leadership incorporates innovation in team and organizational objectives through the integration of processes, policies, and people (Masood et al., 2006; Northouse).

Innovative leadership focuses on open idea exchange, improved capability development, and knowledge sharing (Masood et al., 2006; Northouse, 2010).

"Innovation requires people to think differently, and that thinking can be improved with orderly facilitation" (Miller, 2006, p. 12). Transformational leadership provides the framework for orderly facilitation of innovative thinking within the organization (Northouse). A transformational leader "creates and articulates a vision…provides a role model…provides individualized support…communicates high performance expectations…encourages the acceptance of group goals…[and] provides intellectual stimulation" (Boseman, 2008, p. 37-38). Transformational leaders inspire followers to achieve greater performance and involvement by communicating objectives, increasing trust, encouraging innovation, supporting growth, enabling learning opportunities, fostering understanding, empowering individuals, and setting a positive example (Boerner et al., 2007; Ilies et al., 2006).

Leadership Capabilities

Research conducted by El Emam and Koru (2008) on failed software programs, revealed 28% of respondents believed lack of project management skills for software programs resulted in the cancellation or failure of the project. The study also revealed program failures and cancellation rates are influenced by "organizational maturity, methodology, and project management experience" (El Emam & Koru, p. 89). Leadership capabilities are critical to software engineering program success (Pressman, 2010). Developing and integrating key leadership qualities and approaches can foster improved program performance (Cantor, 2002; Steeneken, 2009). "The long-term success of a company is reliant on management systems that work to foster high-performance and effectively prepare tomorrow's managers and leaders" (Lindbom, 2007, p. 102).

Project management for software engineering is a critical element in determining project success (Cusumano, 2008). Woolridge et al. (2009) noted software program failures are often the result of project management failure to define the software elements for delivery and failure to identify the project problem domain. Software program success is often measured through deviation in schedule, budget, and functionality (Subramanian et al., 2009). Project managers must implement capabilities and processes to define, monitor, and measure project success throughout the life cycle (Sommerville, 2007). Focusing on project management for success, schedule, cost, and product perspectives can assist management in meeting project goals and producing usable products (Subramanian et al.).

Empowerment

In the complex, integrated, and dynamic environment for software development, successful transformational leaders empower team members to meet goals and objectives (Northouse, 2010). The competitive technical environment requires innovative leadership techniques for effectively enabling employee capabilities and motivation (Cagle, 2007). Empowered employees embrace task ownership, exercise self-discipline, and increase efficiency through encouragement and motivation (Chan, Taylor, & Markham, 2008). Empowerment encourages individuals to embrace ownership over the process and accept responsibility for the outcome (Cagle). Leaders empower individuals through elimination of bureaucratic boundaries, limitation of individual activities, and improvement of capabilities (Cagle; Chan et al.).

Empowerment begins with executive leadership and extends to all levels of leadership and individuals throughout the organization (Cagle, 2007). Empowerment provides the framework for implementing individual capabilities through knowledge and motivation (Cagle). In the empowered organization, leaders coordinate activities, acquire resources, plan activities, and coach individuals (Chan et al., 2008). Leaders provide the environment to mentor, coach, train, and facilitate information sharing, self-direction, and autonomy to foster individual growth and development (Chan et al.).

Leaders who embrace empowerment paradigms exchange traditional control and supervision models for practices encouraging support, cooperation, and self-direction (Cagle, 2007). Leaders support empowered employees through information sharing, resource availability, and innovation (Cagle). Individuals are encouraged to embrace innovation and creativity to develop knowledge and enable discovery (Chan et al., 2008). Individuals are empowered and encouraged to be creative, take risks, and learn from mistakes (Chan et al.). In this environment of empowerment and support; innovation, creativity, productivity, and proficiency can flourish (Cagle; Chan et al.).

Communication/Collaboration

Communication and collaboration are essential characteristics of successful software development teams (Mohtashami et al., 2006). Leadership for software engineering initiatives must encourage open communication, provide a collaborative environment, and effectively communicate goals and objectives (Bharadwaj & Saxena, 2006). In the software engineering environment, “better communication between participants can mitigate the drawbacks of diversity by providing a common knowledge base. Hence communication is vital for virtual communities that comprise software developers from diverse backgrounds, in efficiently integrating their knowledge for overall productivity” (Subramanian & Soh, 2008, p. 142).

Communication fosters collaboration, describes organizational tasks, articulates the process, and provides the framework for leadership success (Nielsen, 2009). Leaders interact with employees through communication and collaboration to achieve strategic organizational objectives (Nielsen). Effective leadership communication transmits complex ideas, motivates, inspires, and reinforces ideas (Denning, 2007).

Leaders articulate a clear vision, understand interactions, and stimulate creativity through communication to enable achievement of strategic goals and objectives (Denning, 2007; Nielsen, 2009). The research conducted by Nielsen revealed effective leadership communication was essential for proper interpretation of tasks, learning the vocabulary of the organization, defining reality, correctly interpreting employee expectations, and developing alternative interpretations. As Denning noted, effective leadership communication focuses on conveying the message, stimulating desire, reinforcing with reason, and continuing the conversation to achieve success.

Risk Management

Fully defined risk management approaches incorporate planning, awareness, analysis, mitigation, and monitoring aspects (Dey, Kinch, & Ogunlana, 2007). In the team or distributed environment, risk management should incorporate a focus on effective team collaboration and management into the traditional risk phases (Mohtashami et al., 2006). The most critical aspect for distributed software development is effective management (Mohtashami et al.). In the distributed environment, leadership must factor in the influences of distributed coordination and "define a layered risk management plan to avoid dangers and pitfalls of lack of ownership and authority" (Mohtashami et al., p. 25). This approach will lead to a comprehensive, integrated approach to risk management incorporating the unique components for complex and distributed software engineering environments (Dey et al.; Mohtashami, et al.).

Traditional risk management approaches do not address the unique needs of complex, iterative, and collaborative software development programs (Mohtashami et al., 2006). “Collaborative software development involving multiple organizational units, often spanning national, language, and cultural boundaries, raises new challenges and risks that can derail software development projects even when traditional risk factors are being controlled” (Mohtashami et al., p. 20). Leadership for software development programs must incorporate the unique aspects of distributed development into the risk management process (Dey et al., 2007). Research conducted by Mohtashami et al. noted communication, culture, trust, and rigorous risk management are the critical factors in the collaborative software environment. Development and maintenance of a successful risk management program requires leadership to incorporate practices and processes to address the critical factors for success (Dey et al.; Mohtashami et al.).

The research by Dey et al. (2007) further emphasized the importance of a comprehensive risk management approach. The case study results revealed risk management processes should be an integral component of the life cycle to address risk in the local and distributed software development environment (Dey et al.). Dey et al. observed “effective risk management in software development ensures successful accomplishment of projects with customers’ satisfaction, functional achievement, and overall better financial performance of the organizations” (p. 299).

Leadership for software engineering programs must implement a detailed risk management approach for the software life cycle to address developmental, financial, and organizational risk areas (Dey et al., 2007; Mohtashami et al., 2006). “Managing risk dynamically throughout the project phase will ensure user/customer/client involvement, management commitment, clear specification and design, appropriate planning, realistic expectations, competent and committed staff, and clear vision and objectives” (Dey et al., p. 299). Risk management should be a comprehensive program that integrates the risk process into the collaborative environment (Mohtashami et al.).

Knowledge Management

Knowledge in the corporate world is a critical asset requiring effective management to sustain competitive advantage (Mathew, 2008). Knowledge is used in the software development environment for decision-making, problem solving, and task completion (Mathew & Kavitha, 2008). Research conducted by Landaeta (2008) revealed the performance and capabilities of a project team are increased through knowledge transfer across the organization.

The creativity and complexity associated with software engineering requires information and knowledge sharing to remain current, competitive, and successful (Landaeta, 2008). Effective knowledge management by leadership is a key component of organizational success and supports competitive advantage in the marketplace (Mathew & Kavitha, 2008). According to Landaeta, knowledge provides the foundation and opportunity to improve individual and organizational performance.

For software development organizations, knowledge management is critical to fostering innovative product development, creative design, and continual process improvement (Desouza et al., 2006). Knowledge is critical to the software development life cycle and must be managed in all phases, processes, and practices (Landaeta, 2008).

Leadership must provide team members in the software development environment a robust knowledge development and management program to assure success for project initiatives (Desouza et al.). Effective knowledge management programs collect information on insights, best practices, expertise, and failures to develop a complete knowledge base to promote knowledge reuse for current and future programs (Desouza et al.).

When project teams lack program and organizational knowledge, the risk of not meeting objectives, project cancellation, and program issues is increased (Mathew, 2008; Mathew & Kavitha, 2008). Project leadership must understand the importance of knowledge management and enact knowledge management techniques to collect and disseminate knowledge across projects and the organization (Landaeta, 2008). Research on effective knowledge management for software development reported the critical capabilities for successful knowledge management programs were "strong leadership and having a robust knowledge strategy in operation at all levels" (Desouza et al., 2006, p. 37). In a knowledge environment, project teams obtain the benefits of lessons learned, experiences, and new knowledge development (Mathew & Kavitha). Leadership in knowledge organizations must encourage both informal and formal processes for the exchange and development of critical knowledge for program success (Mathew & Kavitha).

Although knowledge transfer across projects provides positive influences on performance and capabilities (Mathew, 2008), the research conducted by Landaeta (2008) also revealed excessively high knowledge transfer initiatives resulted in negative influences to project performance. Mathew and Kavitha (2008) observed knowledge flow should be frequent but at the proper quantity and quality to foster information sharing and knowledge development. The challenge for leadership is to achieve a balance between information starvation and information overload (Mathew & Kavitha). The key components for organizational success are encapsulated in the leadership practices and methodologies for knowledge development, collection, retention, and dissemination to increase team knowledge and product quality (Fruchter, Swaminathan, Boraiah, & Upadhyay, 2007).

Conclusion

A review of the history of software engineering revealed the complexity and integrated nature of software development presented challenges for successful program completion (Hadar & Leron, 2008; Kirova et al., 2008; Schneidewind, 2007). The development of complex software programs required the use of better methodologies, process, and tools to address the software crisis (Wirth, 2008). As technology and the demand for software applications continued to expand, the need for improved processes for the software engineering life cycle continued to grow (Mahoney, 2008). Although the field of software engineering had developed numerous processes, methodologies, and life cycle models for software development, program efforts have not realized a substantial reduction in failure rates (Cerpa & Verner, 2009; Dalcher & Benediktsson, 2006; Gottesdiener, 2008; Horn, 2009; Mizell & Malone, 2007; Mukherjee, 2008; Pino et al., 2008; Sommerville, 2007; The Standish Group, 2009; Xu & Brinkkemper, 2007).

Software engineering is a complex, dynamic, and human centered effort requiring effective leadership for organizational, product, and process objectives (Adams, 2008; Erdogmus, 2008; Probert et al., 2007). Several research studies investigated the effectiveness of software processes (Agrawal & Chari, 2007; Douglas, 2006; Galin & Avrahami, 2006; Gorry, 2008; Kendall et al., 2008; Shenvi, 2008) or leadership characteristics (Bharadwaj & Saxena, 2006; Bird et al., 2009; Kotlarsky et al., 2007; Nielsen, 2009). In the literature review, a gap exists in research on investigating the effective integrated approach to software engineering. Studies focus on leadership aspects (Dey et al., 2007; El Emam & Koru, 2008; Landaeta, 2008; Mathew & Kavitha, 2008; Mohtashami et al., 2006; Subramanian & Soh, 2008) or methodology aspects (Bose, 2008; Espinosa et al., 2007; Lee et al., 2006; Ramesh et al., 2006) but no studies were found investigating both leadership and process.

This qualitative grounded theory study focused on investigating the leadership approaches and software processes in the open system framework to develop an integrated theory. The primary research question 1 investigated the leadership attributes and the secondary questions, research question 2 and research question 3, focused on software processes. The research provided theory on leadership approaches from the integrated perspective of environment and process.

Summary

The field of software engineering emerged in the 1960s as a result of the need for better methodologies, processes, and tools to address the growing software crisis (Wirth, 2008). As the field of software engineering continued to grow through the 1970s, the concepts of structured approaches to design and development became the standard for formalized methods emphasizing design prior to implementation (Boehm, 2006). In the 1980s and 1990s, the software engineering field concentrated on developing and refining best practices to improve productivity, reliability, and quality (Boehm). In 2000, the software engineering field embraced the concept of continual process improvement to meet the demands of the complex and dynamic software development life cycle (Ebert, 2008). The software development industry realized the need for processes to increase performance and quality throughout the software life cycle (Mahoney, 2008).

The integration of people, tools, and processes fostered continual process improvement and technology development for software engineering (Pressman, 2010; Sommerville, 2007). Software engineering continues to evolve through the implementation and refinement of software life cycle models, techniques, and processes (Pressman; Sommerville). Software life cycle models provide a framework for processes, techniques, and management in a systematic and disciplined approach supporting flexibility to address the unique and dynamic requirements of the software field (Pressman). Methods and approaches to the software engineering life cycle include waterfall development, model driven development, and agile development (Peslak et al., 2008; Pressman).

The software engineering field is changing, growing, and expanding through the introduction of new methods, components, languages, models, and concepts (Boehm & Valerdi, 2008). Software engineering programs and leadership strive to increase productivity and quality while remaining competitive in the technical marketplace (Pressman, 2010; Sommerville, 2007). Software development is a complex, intellectual effort requiring effective leadership for organizational, product, and process objectives (Erdogmus, 2008). Effective leadership requires the application of paradigms to motivate, support, inspire, and foster achievement of individual, project, and organizational objectives (Northouse, 2010; Taylor, 2007).

Chapter 2 presented a review of the research literature on software engineering and software engineering leadership. Chapter 3 presents the methodology of the study and will discuss the appropriateness of the selected methodology. Chapter 3 includes a discussion of the population and sample, sampling methods, a description of the interview instrument, data analysis methodology, reliability, and validity.

CHAPTER 3: RESEARCH METHODS

The purpose of this qualitative grounded theory study was to investigate leadership practices applied to the software development program life cycle to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. A qualitative study was appropriate to collect textual data from participants, ask broad general questions, and analyze responses for themes in a subjective manner (Bogdan & Biklen, 2007; Creswell, 2008; McMillan & Schumacher, 2006). This study examined and developed theories on which leadership practices and processes facilitate successful programs based on observations from software team leaders and team members.

This study explored leadership approaches and software development methodologies to determine the processes which are successful, those which are considered negative impacts to performance, and those which should be updated to enhance program development initiatives. Identification of effective and ineffective processes may lead to continued process improvement to enhance the success rate for software development programs. The identification of successful and unsuccessful leadership approaches for software development activities may provide insight into appropriate methods to integrate into the software process to foster improvement in software leadership paradigms.

Chapter 3 provides a discussion of the selected research method and research design. The appropriateness of the qualitative methodology and grounded theory design is presented along with a discussion of why other methods would not meet the objectives of the study. The study research questions, sampling frame, population, data collection approach, instrumentation, validity, reliability, and data analysis procedures is also presented.

Research Method and Design Appropriateness

Software is an integral part of products for consumers, business, government, and the military (Basili et al., 2008; King, 2007; Probert et al., 2007). Although software programs continue to grow in complexity and criticality, the success rate for development programs has not improved (Cerpa & Verner, 2009; Horn, 2009; The Standish Group, 2009). Research revealed failure rates for software programs are increasing whereas success rates are decreasing (Rubinstein, 2007; The Standish Group). The software life cycle depends on the implementation of development methodologies, process, and leadership approaches for success (Agrawal & Chari, 2007; Sapienza, 2005; Tesch et al., 2007). The study investigated the perceptions of software development leaders and team members to identify the processes and approaches contributing to successful program development.

The study implemented a qualitative grounded theory design to obtain information on the perceptions, views, and opinions of software team members and leadership. An open-ended electronic interview questionnaire was provided to participants to explore the perceptions, views, beliefs, and attitudes related to software life cycle leadership processes. "Qualitative data collection consists of collecting data using forms with general, emerging questions to permit the participant to generate responses; gathering word (text) or image (picture) data; and collecting information from a small number of individuals or sites" (Creswell, 2008, p. 213).

Qualitative research emphasizes context and the relation of themes and trends based on the circumstances for the topic under study (Creswell, 2008; McMillan & Schumacher, 2006; Salkind, 2003; Schram, 2006; Shank, 2006). The qualitative study methodology supported collecting observations and results on the application of various software development processes from leadership and team members. The data collected from study participants was obtained by developing general open-ended questions on software engineering methodologies, approaches, and processes. The study responses were analyzed to identify trends, themes, characteristics, and behaviors supporting successful software development programs.

Qualitative research focuses on meaning and understanding through investigation in the subject environment (McMillan & Schumacher, 2006). Qualitative research does not restrict the views and perceptions of the participants but seeks to gain understanding and identify theories through open responses (Creswell, 2008; Leedy & Ormrod, 2005; Salkind, 2003; Shank, 2006). Qualitative research is a systematic "form of inquiry that depends upon the world of experience in some fundamental way" (Shank, p. 5). The study explored the perceptions of software engineering leadership and team members through collecting open-ended responses to an electronic interview questionnaire. The use of open-ended electronic interview questionnaires "provides rapid access to large numbers of people and a detailed, rich database for qualitative analysis" (Creswell, p. 227).

The research organization selected for the study consisted of distributed team members in multiple locations. The use of the electronic questionnaire provided access to a larger sample and allowed input across all team member functions and levels of expertise. McMillan and Schumacher (2006) noted in face-to-face interviews, the researcher's presence can often influence responses. The use of electronic open-ended questions allowed maximum flexibility in participant responses (Creswell, 2008) and provided a rich data set for review and analysis without constraining individual responses (Creswell; Shank, 2006). Open-ended questions supported participant responses from individual cultural, social, and technical experiences and did not limit the responses to researcher views or perceptions (Creswell; Leedy & Ormrod, 2005; McMillan & Schumacher; Schram, 2006; Shank).

Berg (2009) observed the use of the computer for communication and interaction has become comfortable and common in technical environments. The use of computer assisted interviews for qualitative research can transfer the comfort and familiar feel of technology usage to the interview process (Berg). The electronic interview questionnaire provides benefits over the face-to-face interview such as avoiding impacting demanding work schedule or job requirements, providing an unambiguous context for responses, incorporating a common and familiar method of communication, and avoiding errors in transcription of responses (Berg; Beadell, 2009).

Beadell (2009) conducted a qualitative study on software process improvement implementing an electronic interview questionnaire. Beadell observed the electronic interview questionnaire provided advantages in collecting data by allowing participants to "respond unobtrusively when they had spare time, either before or after work, and afforded them a focused opportunity to be candidly objective in the quietness of their office" (p. 103). The electronic interview also maximized the effectiveness and efficiency of collecting and transmitting data, improved the data accuracy without recording or transcription errors, and provided responses full of elaborations and examples (Beadell). Beadell further observed that the candid and rich responses might not have been acquired in a face-to-face structured interview.

As noted by Creswell (2008), quantitative research focuses on measuring variables and differences in variables for two or more groups. Qualitative research does not strive to measure or compare groups or variables but strives to reach an expanded understanding of the perceptions and views of an individual or single group of individuals (Creswell; McMillan & Schumacher, 2006; Schram, 2006). Quantitative research focuses on measuring the typical or ordinary condition (McMillan & Schumacher; Shank, 2006). Qualitative research seeks to investigate conditions and settings not functioning in the usual manner (McMillan & Schumacher; Shank). The qualitative study investigated the "settings and conditions where things are not operating as usual" (Shank, p. 106).

A qualitative study was appropriate to collect textual data from participants, ask broad general questions, and analyze responses for themes in a subjective manner (Bogdan & Biklen, 2007; Creswell, 2008; Leedy & Ormrod, 2005; McMillan & Schumacher, 2006; Schram, 2006; Shank, 2006). This study sought to explore and understand the leadership processes for successful program development. The qualitative research study was appropriate to investigate the views and perceptions of software development leadership to identify trends and themes. The qualitative method explores participant views of a central phenomenon to enhance understanding, identify themes, and develop theories (Bogdan & Biklen; Creswell; Leedy & Ormrod; McMillan & Schumacher; Schram; Shank).

The open-ended electronic interview questionnaire allowed respondents to provide thoughts, views, and perspectives on software engineering and leadership processes. The electronic interview provided access to large numbers of individuals to obtain a rich textual database for qualitative analysis (Creswell, 2008). The results obtained were reviewed to identify trends, themes, and concepts on successful and unsuccessful software engineering practices.

Qualitative research methods investigate phenomenon by analyzing textual data, graphic data, or observations (Creswell, 2008; Schram, 2006). The qualitative method provides several research designs for developing the study and collecting data, including phenomenological, case study, ethnographic, and grounded theory (Leedy & Ormrod, 2005; McMillan & Schumacher, 2006; Salkind, 2003; Schram). Each of these designs was examined for use as the methodology for this study. After examination of each of these designs, the grounded theory methodology was selected as the most appropriate qualitative research design to match the research purpose and question.

The phenomenological methodology was reviewed as a possible research design for this study. Phenomenological research designs focus on investigating individual awareness and experiences through narratives and descriptions (Schram, 2006). Phenomenology seeks to attach meaning to a phenomenon under study by investigating the experiences of individuals (Leedy & Ormrod, 2005).

The phenomenological research design does not support development of theories based on collected data but seeks to investigate a concept or phenomenon (Schram, 2006). This research study seeks to develop a theory on leadership processes that contribute to successful software development programs. The phenomenological research method was not suited to meet the intent of the study. The grounded theory research design seeks to enhance current theory and develop theories based on collected data from social research (Glaser & Strauss, 1967, 2007; Leedy & Ormrod, 2005; Schram).

The case study methodology is “an analytic focus on an individual event, activity, episode, or other specific phenomenon” (Schram, 2006, p. 106). This study seeks to investigate the leadership attributes that contribute to successful software engineering initiatives through the analysis of several development programs and not an individual case. Implementing a case study approach for this research was not appropriate because a case study focuses on describing individual actions or human behavior around a central event or phenomenon (Leedy & Ormrod, 2005; Salkind, 2003; Schram).

This research study investigated the beliefs, attitudes, values, and behaviors of software engineering leaders to develop a theory for leadership of successful software development programs. Grounded theory research attempts to identify patterns and themes in the collected data to understand the processes and develop theories (Charmaz, 2006; Leedy & Ormrod, 2005). The grounded theory design provides a rich data set for analysis and theory development (Leedy & Ormrod).

The ethnographic methodology focuses on developing meaning of phenomena through the observation of a group of individuals (Creswell, 2008; Schram, 2006). Ethnographic research designs describe and interpret a cultural group’s beliefs, behavior, and patterns that develop (Creswell; Leedy & Ormrod, 2005). Ethnographic designs strive to reveal common cultural understandings based on the observation of a common group (Schram). The ethnographic design was not appropriate because the intent of the study was not to reveal common understandings in software development, but rather to develop theories on the leadership processes for success.

Corbin and Strauss (2008) observed grounded theory supports the development of innovative views and theories. The qualitative grounded theory methodology supports enhancement of current theories and introduction of new theories based on the observations and experiences of participants (Glaser & Strauss, 2007; Leedy & Ormrod, 2005). Grounded theory does not focus on testing existing theory (Creswell, 2008; Schram, 2006). Grounded theory focuses on an area of investigation and allows theories to develop and emerge from the data (Creswell; Glaser & Strauss, 1967, 2007; Schram).

Research Questions

This qualitative grounded theory research study focused on leadership practices for software engineering programs. The research question focused on the investigation of leadership impacts for program development. Research question 1: What leadership characteristics contribute to the positive outcome of software development programs? The supplemental research questions focused on software practices. Research question 2: What software practices contribute to the development of successful projects? Research question 3: What software practices result in negative impacts to projects? These research questions will guide and focus the study on the investigation and identification of processes and practices that lead to successful software development programs and the impacts of leadership approaches.

Woolridge et al. (2009) observed proper leadership planning and management may decrease the project failure rate by providing effective scope definition approaches. Subject matter experts in the field of software engineering have investigated the effectiveness of software processes (Boehm, 2006; Kenett & Baker, 2010). Additionally, scholars have researched improvements in processes for software development (Beadell, 2009; Bonner, 2008; Sun, 2008) or improvement in leadership capabilities (Early, 2006; Jain, 2007; Johnson, 2008). The research questions for this study focused on effective integration of leadership capabilities and software processes for success. A grounded theory study focusing on leadership and process added to the existing body of knowledge by integrating the leadership and process dimensions for software engineering.

Population

Qualitative research seeks to identify "participants and sites based on places and people that can best help us understand our central phenomenon" (Creswell, 2008, p. 213). This study focused on leaders and team members of software development programs at a research organization in Alabama. The study invitation was distributed to 600 members of the selected organization. Respondents to the study invitation were asked several demographic questions to identify individuals with experience in software engineering and leadership. Data was collected from individuals meeting the sampling criteria.

Sampling Frame

The study was conducted at a software research and development facility in Alabama. The organization selected had significant experience in developing software applications, obtained a level four CMM rating, and had numerous on-going development efforts. The organization represented a typical software development facility with distributed software teams. This environment provided a depth of information and expertise that produced a rich data set for evaluation.

A study invitation was provided to the organization e-mail distribution list. Participation in the research study was voluntary. The study invitation included a link to the interview questionnaire for those individuals who elected to participate. To focus on leadership for software development, demographic questions were included in the questionnaire to collect responses from leaders and team members of software development programs. The questionnaire included skip logic for the demographic questions to exit the survey for any respondents not in the desired sampling frame. Responses were not collected from individuals in the organization not involved in software development programs.

The goal of grounded theory research is to obtain enough data to identify patterns, themes, concepts, and perceptions focusing on the central phenomena (Creswell, 2008; Glaser & Strauss, 2007; Neuman, 2003). The sampling frame in grounded theory must generate enough data for analysis (Corbin & Strauss, 2008; Creswell; Glaser & Strauss, 1967, 2007). For this study, theoretical sampling was used to select participants with knowledge in software development and leadership. These experts in software engineering provided the best available data for analysis and theory development (Bogdan & Biklen, 2007; Corbin & Strauss; Creswell; Leedy & Ormrod, 2005; Neuman; Shank, 2006).

The study invitation was provided to the organization population of 600 employees. The demographic questions were used to select the individuals with experience in software engineering and leadership. The number of participants was 12% of the invited population or 71 respondents. Creswell (2008) indicated a sample of one to 40 participants is acceptable for qualitative research. Larger sample sizes can result in difficulty managing data and subjects and may produce superficial perspectives (Creswell). In qualitative research, the ability to acquire detailed information decreases as the number of study participants increases (Creswell). Suzuki et al. (2007) also noted 3 to 30 study participants is an acceptable sample size for qualitative research to produce meaningful results.

The study included periodic reminders to the population to increase the response rate. Qualitative research data collection typically continues until saturation has occurred (Creswell, 2008; Schram, 2006; Shank, 2006). Saturation occurs when the research is no longer producing new findings or the investigation has reached the point in which the same data is being collected (Creswell; Schram). Saturation is much like the law of diminishing returns; the repetition in responses indicates additional research will not produce further new, significant, or unique information (Creswell; Schram; Shank). For this study, the time constraints for study completion, limited sample size, response rate, and participant suitability affected the amount of data collected.

Saturation results when the responses are not generating any additional findings (Glaser & Strauss, 1967). Bogdan and Biklen (2007) observed in qualitative research, the researcher can collect too much data and must define a finishing point for data collection. Continuing to collect data past this point will result in diminishing returns for the time spent collecting and analyzing additional data (Bogdan & Biklen).

In grounded theory research, small sample sizes are common and do not prevent the formulation of theories grounded in the raw data (Creswell, 2008; Suzuki et al., 2007). Due to the time constraints for the study, a static sampling method was used in which all data was gathered prior to analysis (McCleaf, 2007; Polkinghorne, 2005).

Polkinghorne observed static sampling does not support data gathering from additional participants but is often practical for data gathering.

For the study, the researcher did not seek data saturation but rather a finite and rich data set for analysis. As Creswell (2008) observed, small sample sizes in grounded theory research do not prevent the formulation of theories grounded in the raw data. The small sample sizes often result in a rich data set for analysis and evaluation (Creswell; Suzuki et al., 2007).

The researcher reviewed the collected data for common themes, perspectives, and views. The final data coding and analysis was conducted for all respondents who met the sample criteria of involvement in software development programs. Permission was obtained to conduct the study and access the e-mail distribution list (see Appendix A) at the selected research and development facility.

Informed Consent

A study invitation (see Appendix B) was provided to the e-mail distribution list. The study invitation included a description of the study purpose, assurance that the questionnaire responses were anonymous, and a link to the questionnaire website. The initial page of the questionnaire website included a description of the study purpose and assurance that participation was anonymous. Study participants provided their informed consent to participate (see Appendix C). Respondents not willing to consent to the study were able to exit the website without participating or providing any answers to the interview questions.

Participants had a second opportunity to withdraw from the study. At the completion of the questionnaire, respondents were required to select whether to submit their responses or exit for the website without participating. Previously entered responses were deleted for individuals selecting to exit without participating. Data was not collected or stored for any participant who elected to exit the website without submitting the questionnaire.

Confidentiality

The participant identities and responses remained confidential in the research process and report. The researcher maintained and controlled the information provided by the study participants in a secured area. During review and analysis, a coding process was implemented to provide additional assurance of participant anonymity. Data collected electronically was downloaded to removable storage media for archiving.

Research information and documentation collected during the study process will be maintained in a secured container for 3 years. The secured container is located in the researcher's office in a controlled access facility. After 3 years, all study hard copy and electronic media will be destroyed using appropriate methods.

Geographic Location

The location of the study was the state of Alabama. The specific institution participating was not disclosed due to possible confidentiality concerns. The institution was a software research and development organization in Alabama. The organization selected specializes in the development of complex, embedded software programs through distributed software development teams.

Data Collection

This qualitative study used an electronic open-ended interview questionnaire as the primary instrument for data collection (see Appendix D). The interview questionnaire contained basic demographic questions to determine the applicability of respondents for the study. The demographic questions assured the respondents have been involved in software development programs, identified the level of expertise, and identified the team member role.

The open-ended questions focused on investigating the views, beliefs, and perspectives related to software engineering development life cycle processes and leadership approaches. The qualitative data collection approach supported general, broad questions to participants to obtain unrestricted perspectives. The open-ended electronic interview allowed respondents to provide thoughts, views, and perspectives to obtain a rich textual database for qualitative analysis (Creswell, 2008).

The electronic questionnaire consisted of four sections; assurance of confidentiality and informed consent, closed-ended demographic questions, open-ended questions on software development, and a final opportunity to withdraw from the study without submitting responses. Participation in the study was voluntary. The invitation included a link to a questionnaire website. Data collected on the website was maintained in a confidential, password protected database accessed by the researcher. At the conclusion of the participant response timeframe, the data collected was downloaded from the website and stored on removable media. All information system components used for data collection and analysis were access controlled and password protected.

Instrumentation

The research study used an electronic interview questionnaire as the instrument. The questionnaire consisted of open-ended questions on software development and closed-ended demographic questions. The open-ended interview questions allowed the study participants to share experiences, views, and perceptions on leadership and software engineering. Implementing a closed-ended questionnaire would not allow the participants to share perceptions and experiences and would not produce the depth of information required to identify themes and trends (Creswell, 2008; Shank, 2006).

Creswell (2008) observed an acceptable method of data collection for qualitative research is to “collect open-ended responses to an electronic interview or questionnaire” (p. 221). Electronic interviews are effective for collecting data from a dispersed group of individuals. The electronic interview provides access to numerous potential participants and can provide rich and detailed data for qualitative analysis (Creswell). Babbie (2010) observed the electronic self-interview is an acceptable method of data collection where the participant completes the questionnaire using the Internet and the results are provided to the researcher for review and analysis.

Persichitte, Young, and Tharp (1997) noted several advantages to electronic interviews over face-to-face interviews. Advantages of electronic interviews include allowing respondents to be more careful and thoughtful in responding, elimination of scheduling conflicts, reduced interruptions, and simplified data recording (Persichitte et al.). The use of the electronic interview provides the participant with the flexibility to review questions, reflect on responses before entering, and complete the interview within individual time requirements (Babbie, 2010; Persichitte et al.).

The electronic interview has been successfully implemented in several qualitative research studies. McCleaf (2007) collected data through an electronic questionnaire for a qualitative grounded theory study on achievement of academic success for minority females. Ogle (2009) implemented an e-mail questionnaire to collect data for an exploratory qualitative study on hotel management and customer satisfaction. In research on positive development of youth through sports participation, Greenwood and Kanters (2009) implemented a web-based self-administered questionnaire to collect data for qualitative analysis.

In a research study exploring e-learning problems and solutions, Fichten et al. (2009) implemented an online questionnaire consisting of closed-ended demographic questions and open-ended interview questions. Baillie, Ford, Gallagher, and Wainwright (2009) collected data for research on dignity in health care for the elderly using an Internet questionnaire with both fixed response and free text questions. These studies all collected data for qualitative analysis through the implementation of an electronic interview questionnaire.

The purpose of this qualitative grounded theory study was to investigate leadership and development practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. A qualitative study was appropriate to collect textual data from participants, ask broad general questions, and analyze these responses for themes in a subjective manner (Creswell, 2008; Shank, 2006). The interview questionnaire provided questions to investigate participant views on the leadership characteristics and software development processes considered successful and unsuccessful.

Developing and implementing successful projects requires the effective balancing of resources, environment, and processes (Cusumano, 2008). The theoretical framework for this study focused on the open system paradigm integrating leadership approaches to resources, environment, and processes (National Defense University, 2009; Scott &

Davis, 2007). The interview questionnaire investigated software development in the framework for open systems by developing questions on leadership capabilities and software processes. The leadership capabilities explored the environment and approach applied to software development programs. The process focused questions investigated the processes and resources considered successful and unsuccessful in software development initiatives.

The interview questionnaire implemented open-ended questions to obtain participant perceptions, beliefs, and views (Creswell, 2008; Shank, 2006) on leadership and processes for software development. Questions on software leadership characteristics were developed to address the primary research question 1: what leadership characteristics contribute to the positive outcome of software development programs? Interview questions on successful and unsuccessful software processes were developed to explore the supplemental research questions on software practices.

Several grounded theory research studies have developed interview questions (Beadell, 2009; Bixenman, 2007; McCleaf, 2007; Murray, 2008) to investigate views and perceptions of the participants for a particular phenomenon (Creswell, 2008; Neuman, 2003). In these studies, the type and number of questions varied but each study implemented three to five open-ended questions on the primary phenomenon under study. Beadell developed four questions related to CMMI compliancy impacts, Bixenman developed three questions related to leadership for innovation, McCleaf developed four questions related to minority academic success, and Murray developed five questions related to leadership characteristics. The studies by Beadell, McCleaf, and Murray also included a final question allowing the participants to provide any additional views and perceptions.

This grounded theory research study implemented four primary questions on leadership for successful software development programs. The electronic interview questionnaire contained four open-ended questions focusing on leadership for software development and a final question allowing participants to provide additional thoughts, views, and perceptions. The questionnaire contained demographic questions and three open-ended questions focusing on the secondary topic of software processes.

The researcher obtained participants from the organization e-mail distribution list. A study invitation was distributed to the e-mail list provided and included a description of the study and a link to the anonymous questionnaire website. The interview questionnaire contained four sections; informed consent, demographic questions, interview questions, and opportunity to withdraw for the study.

The initial section contained an assurance of confidentiality and informed consent. This section informed participants of the study purpose and data collection process. Participants provided their consent to participate or elect to exit from the questionnaire without participating. Participants completing the informed consent and electing to participate continued to section 2.

The second section contained closed-ended demographic questions to assure participants meet the study selection criteria. This section included skip logic to exit the survey for any respondents not in the desired sampling frame. Responses were not collected from individuals in the organization who do not meet the sampling criteria. If an individual did not meet the sampling criteria, they were thanked for participation and exited from the interview.

The third section consisted of open-ended questions on software development and leadership processes. This section implemented questions focused on exploring perceptions of leadership attributes and development processes considered successful and unsuccessful. The participants were provided with the opportunity to provide any additional comments or information on leadership and software development.

The fourth section provided an opportunity for the participant to withdraw from the study. Participants only reach this section after completing the informed consent, section 2, and section 3. In section 4, the participant could elect not to participate after completing the questionnaire and the informed consent. If the participant selected not to participate, data entered in the previous sections was deleted and not collected for analysis. If the participant selected to participate, the previously entered data was collected and stored for analysis.

The interview questionnaire was developed by the researcher to address the unique areas for investigation. No study instrument existed to investigate software development and leadership processes for theory development. A unique interview questionnaire was required to support data collection. The questionnaire was verified through a pilot study. The completion of the initial pilot study assisted in verifying the functionality of the selected questionnaire technology, verifying the accessibility of the questionnaire website, and verifying the clarity and focus of the qualitative questions on the desired study topics.

Schram (2006) noted pilot studies are essential to understanding concepts, identifying researcher assumptions and biases, and verifying the research method and instrument. The pilot study provided the study instrument to five participants to obtain feedback on study approach, format, and question phrasing. The pilot study responses were not included in the data collection. The pilot study participant views were reviewed and evaluated to improve the questionnaire quality. The research instrument was updated as required from the observations, feedback, and analysis from the pilot study.

Validity and Reliability

Validity is concerned with assuring the data and information collected reflects accurate facts and truth (Shank, 2006). The research study validity was dependent upon the study participants providing truthful responses to the questionnaire (Shank). The validity of the data analysis was dependent upon adequate assessment of the textual responses by the researcher (Neuman, 2003; Shank).

In qualitative research, validity focuses on authenticity, or providing an accurate representation of social phenomenon from the participants view (Neuman, 2003). Quantitative researches focus on matching concept to data, but qualitative research focuses on providing an accurate portrayal of the experiences of the participants (Neuman). Validity for qualitative researchers focuses on providing an accurate representation of the participants and events under study (Neuman).

Internal validity is concerned with errors internal to the design of the research (Neuman, 2003). Internal validity focuses on possible errors in data collection and analysis that could affect the study results (Neuman). External validity focuses on the ability to generalize the findings to a broader range of environments and individuals

(Neuman). In qualitative research, results that “can be generalized to many situations and many groups of people” (Neuman, p. 187) have high external validity. Results that “apply only to a very specific setting” (Neuman, p. 187) have low external validity. For this study, the results were only generalized to the specific setting of software development environments.

Reliability focuses on assuring the accuracy of the observations, information, and data collected (Shank, 2006). The electronic questionnaire eliminated some of the issues associated with data transcription and data entry. The participants provided responses to each question. The reliability of the data was dependent upon the respondent providing accurate and truthful responses (Shank). The reliability of the data can also be influenced by the analysis (Neuman, 2003; Shank). One method to prevent researcher bias being introduced into the analysis assures the coding used to group themes and trends during data analysis is consistent and accurately reflects the participant responses (Shank).

To assure accurate and dependable data collection, a pilot study of the electronic questionnaire was conducted before the research study. The pilot study verified accessibility of the questionnaire website, the functionality of the selected questionnaire technology, and the quality of the open-ended questions. The pilot study also verified the clarity and focus of the study questions on the desired study topics. Pilot studies are essential to understanding concepts, identifying researcher assumptions and biases, and verifying the research method and instrument (Schram, 2006).

Data Analysis

The qualitative study used an open-ended questionnaire to collect textual data from participants and ask broad general questions to analyze the responses for themes

(Creswell, 2008). The questionnaire obtained views and perceptions from software program leaders and team members to identify trends, behaviors, and characteristics for successful management of software programs. A questionnaire was appropriate when attempting to investigate and describe views, beliefs, attitudes, or aspects for a particular group (Creswell). This study questionnaire explored leadership approaches and software development methodologies to determine the processes which are successful, those which are considered negative impacts to performance, and those which should be updated to enhance program development initiatives. The study responses were analyzed to identify trends, themes, characteristics, and behaviors supporting successful programs.

In qualitative research the data must be reviewed, organized, and analyzed to reflect the perceptions of the participants (Shank, 2006). The data coding process sorts the data into broad categories based on concepts, patterns, or similar features for analysis by the researcher (Neuman, 2003; Shank). After the data is sorted or coded, the data is again reviewed to explore what themes or patterns can be identified (Shank).

Neuman (2003) observed "qualitative coding is an integral part of data analysis" (p. 441). For grounded theory research, data coding can be performed in several stages to refine and manage the large amount of data collected (Creswell, 2008; Neuman). The three stages of data coding that can be applied in grounded theory research are open coding, axial coding, and selective coding (Corbin & Strauss, 2008; Neuman).

Open coding is the first review of the data by the researcher to sort the volume of data into manageable themes or codes (Neuman, 2003). The information is segmented into categories based on the phenomenon under study (Creswell, 2008). Axial coding is a second pass through the data sorting the data based on the codes identified in the first review of the data (Neuman). In axial coding, the researcher refines and links themes and concepts for the data (Creswell; Neuman). Selective coding involves reviewing data and codes to refine and identify major themes and concepts in the data (Neuman).

In grounded theory research, Corbin and Strauss (2008) defined thematic analysis as a process for coding and analyzing data. Thematic analysis focuses on identifying patterns and themes in the data (Shank, 2006). As common observations and views are identified in the data, themes and patterns emerge grounded in the raw data (Shank).

The textual data collected was analyzed for common trends, themes, and views. NVivo 8 (QSR International, 2008) qualitative data analysis software was employed to support data management, tracking, and organization. Collected data was downloaded into NVivo 8 to effectively sort, track, and evaluate the raw data. Data was tabulated to collect the common perceptions and trends provided in the questionnaire responses. Analysis of the textual data provided in-depth meaning and understanding on the concepts for the phenomenon under study (Creswell, 2008; Neuman, 2003).

The data analysis included review of the responses for common themes, words, and concepts provided by the study participants. Themes were categorized based on key leadership concepts, software processes, and open system theory. Data was analyzed to identify theory and methods on successful leadership and process applications for successful software development. Results of the data analysis supported theory development on capabilities, themes, processes, and concepts for successful software development and leadership.

Summary

The purpose of this qualitative grounded theory study was to investigate leadership practices applied to the software development program life cycle to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. This study examined and developed theories on which leadership practices and processes facilitate successful programs based on observations from software team leaders and team members. The research study implemented a qualitative research grounded theory approach to obtain information on the perceptions, views, and opinions of software team members and leadership.

An open-ended electronic interview questionnaire was provided to participants to explore the perceptions, views, beliefs, and attitudes related to software life cycle leadership processes. Qualitative research for this study was appropriate to emphasize context and the relation of themes and trends based on the circumstances for the topic under study (Schram, 2006). Qualitative research does not limit the responses and view of the participants but attempts to gain understanding and identify theories through open responses (Creswell, 2008; Shank, 2006).

This study focused on leaders and team members of software development programs at a research organization in Alabama. The study invitation and informed consent was distributed to all members of the organization on the e-mail distribution list. The study was distributed to approximately 600 individuals with theoretical sampling methodologies for data collection. The organization selected had significant experience in developing software programs for multiple applications and was representative of a typical software development facility with distributed software teams. The selected organization provided access to a depth of information and expertise on software development to produce a rich data set for evaluation.

A study invitation was provided to the e-mail distribution list providing a description of the study purpose, informed consent, notice that participation is voluntary, assurance that the questionnaire responses were anonymous, and a link to the interview questionnaire website. The initial page of the interview website included a description of the study purpose and asked respondents for consent to the study. Data was not collected or stored for any participant who elected to exit the website without submitting the questionnaire. The participant identities and responses remained confidential during the research process and report generation. Information provided was stored in a secured area controlled by the researcher and will be destroyed after 3 years.

An electronic interview questionnaire was the primary instrument for data collection. The interview questionnaire contained four sections encompassing assurance of confidentiality and informed consent, closed-ended demographic questions, open-ended questions on software development, and a final opportunity to withdraw from the study before submitting responses. The demographic questions assured the respondents had been involved in software development programs. The open-ended questions focused on investigating the views, beliefs, and perspectives related to software engineering development life cycle processes and leadership approaches.

The qualitative research data collected was reviewed, organized, and analyzed to reflect the perceptions of the participants (Shank, 2006). The data coding process used open coding, axial coding, and selective coding to organize and analyze the data (Corbin & Strauss, 2008; Neuman, 2003). Open coding provides an initial sort of the data, axial coding reviews and sorts the data based on the codes identified, and selective coding identified major themes and concepts (Creswell, 2008; Neuman). As common observations and views were identified in the data, themes and patterns emerged grounded in the raw data (Shank).

The textual data collected was analyzed for common trends, themes, and views. The data was analyzed to identify and provide in-depth meaning and understanding on the concepts for successful software development programs. The data analysis included review of the responses for common themes, words, and concepts for categorization based on key software concepts. Results of the data analysis supported theory development on software engineering leadership by identifying the trends, themes, and concepts on successful and unsuccessful software engineering practices. Chapter 4 presents a discussion on the data collection process, data analysis procedures, and results of the study.

CHAPTER 4: RESULTS

The purpose of this qualitative grounded theory study was to investigate leadership and development practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. The goal of this study was to explore and analyze leadership and development processes for software engineering to identify a grounded theory of characteristics that result in successful software development programs. The research explored the experiences and unique perceptions of leaders and software developers actively involved in software development programs.

Chapter 4 presents the results of the research study. An overview of the data collection and analysis process is presented. This chapter provides a description of the pilot study, pilot study results, research study, and themes identified.

Findings

This qualitative grounded theory research study investigated the software engineering leadership processes and approaches that contribute to successful software programs. Data was collected using an Internet questionnaire to obtain view, perceptions, and beliefs from software engineering experts and team members. Data collection and analysis were performed as described in Chapter 3. The sections of this chapter provide an overview of the data collection and analysis, pilot study and results, instrumentation, study population, and themes identified.

Data Collection

This qualitative study implemented an electronic open-ended interview questionnaire as the primary instrument for data collection (see Appendix D). The interview questionnaire contained basic demographic questions to determine the applicability of respondents for the study. Respondents without experience in software development or leadership were exited from the interview. The open-ended interview questions focused on investigating the views, beliefs, and perspectives related to software engineering development life cycle processes and leadership approaches. The electronic questionnaire contained four sections; assurance of confidentiality and informed consent, closed-ended demographic questions, open-ended questions on software development, and a final opportunity to withdraw from the study without submitting responses.

This study questionnaire explored leadership approaches and software development methodologies to determine the processes which are successful, those which are considered negative impacts to performance, and those which should be updated to enhance program development initiatives. The study responses were analyzed to identify trends, themes, characteristics, and behaviors supporting successful software programs. A data coding process was used to sort the data into broad categories based on concepts, patterns, and features for analysis. After data sorting, the data was reviewed a second time to identify themes and patterns. As common observations and views were identified in the data, themes and patterns emerged for development of grounded theory.

The data collected was analyzed for common trends, themes, and views. The raw data collected was sorted for evaluation and analysis. Themes were categorized based on key leadership concepts and software processes. Results of the data analysis and theme identification were used for theory development on capabilities, themes, processes, and concepts for successful software development and leadership.

Pilot Study

The electronic interview questionnaire was developed for this research study. Because the questionnaire was new, a pilot study was conducted to verify the adequacy of the distribution method, interview questions, and skip logic for demographic questions. The pilot study was conducted during the first three weeks of May 2010. Five individuals with specific knowledge of software engineering leadership were selected to participate. Three of the individuals had 10 or more years experience and were project leaders. The other two individuals had less than 10 years experience and were software developers.

The pilot study participants were provided a link to the interview questionnaire. Participants were asked to complete the interview as presented. Participants were then asked to provide feedback on the adequacy of the electronic format, interview questions, and suggestions for improvement. Each of the five individuals completed all interview questions and provided feedback for improvement of the questionnaire.

The primary goal of the pilot study was to determine if the interview questions approximated the intent for the research study. The demographic and open-ended questions were completed by each of the five participants. Review of the responses revealed that no changes were required for the demographic or open-ended interview questions.

A secondary goal of the pilot study was to determine the adequacy of the data collection method and procedures. The participants were asked to provide feedback on the adequacy of the electronic method and approach. The five participants stated the electronic questionnaire was easy to use and found no problems in accessing the questionnaire or entering responses.

The respondents provided two suggestions on the questionnaire format that were incorporated into the final interview questionnaire. The first suggestion was to add a progress indication on each page of the survey. The second suggestion was to present one open-ended question per page. The pilot study presented all open-ended questions on one page. The participants suggested this format presented too much information on one page and could result in a lower response rate. The suggestions were incorporated into the final electronic interview questionnaire.

Instrumentation

The instrument for this research study was an electronic interview questionnaire created for this study. The questionnaire consisted of closed-ended demographic questions and open-ended questions on software development and leadership. The demographic questions obtained information on years of experience in software engineering, position, and years of experience in leadership. The open-ended questions allowed participants to share experiences, views, and perceptions on leadership and software engineering. The electronic interview allowed collection from a distributed group of individuals and supported participation by numerous individuals.

The interview questionnaire contained four sections; informed consent, demographic questions, interview questions, and opportunity to withdraw for the study. The initial section contained an assurance of confidentiality and informed consent. Participants completing the informed consent and electing to participate continued to section 2. The second section contained closed-ended demographic questions to assure participants met the study selection criteria. Skip logic was implemented in this section to exit the questionnaire for respondents not in the desired sampling frame.

The third section consisted of open-ended questions on software development and leadership processes. The final section provided an opportunity for the participant to withdraw from the study. If the participant selected not to participate, data entered in the previous sections was deleted and not collected for analysis. If the participant selected to participate, the previously entered data was collected and stored for analysis.

Study Population

A study invitation (see Appendix B) was provided to a research and development organization e-mail distribution list in Alabama with a population of 600 employees. Participation in the study was voluntary. The study invitation included a description of the study purpose, assurance that the questionnaire responses were anonymous, and a link to the questionnaire website. Study participants provided their informed consent to participate (see Appendix C) on the first page of the questionnaire. Respondents not willing to consent to the study were able to exit the website without participating or providing any answers to the interview questions.

The study invitation was distributed to the organization population on 1 June 2010. The study was open for data collection until 30 June 2010. At the end of the data collection period 158 respondents had accessed the questionnaire. Of the 158 respondents to the questionnaire, 142 agreed to the informed consent. Any participant without experience in software development or leadership of software development programs was exited from the study. Additionally, some participants completed the demographic portion but did not provide responses to any of the open-ended questions. Respondents not answering any of the open-ended questions were removed from the study sample for analysis. Data from the remaining participants was collected for analysis. Data was collected for analysis from the respondents agreeing to the informed consent, answering the demographic questions, meeting the sampling criteria, providing answers to one or more of the open-ended questions, and electing to submit answers in section 4 of the questionnaire. Table 1 provides an overview of the participants for the study.

Table 1

Respondent Overview

Category	Number
Respondents to Study Invitation	158
Respondents Agreeing to Section 1 Informed Consent	152
Respondents Answering Demographic Questions	142
Respondents Answering Open-ended Questions	72
Respondents Agreeing to Section 4 Informed Consent	71

As shown in Table 1, 158 individuals responded to the study invitation (see Appendix B). After elimination of the individuals not agreeing to the informed consent, individuals not meeting the sampling criteria, and individuals not answering any of the open-ended questions, 71 participants provided data for the study.

Table 2 provides an overview of the participant job functions. All participants selected job functions related to leadership or software engineering. Individuals selecting the other category reported job functions of software analyst, information assurance, configuration management, and software training. These individuals were included in the survey.

Table 2

Participant Information on Job Function

Job Function	Frequency	Percentage
Executive Leadership	4	5.6
Team Leader	20	28.2
Senior Software/System Engineer	20	28.2
Software/System Engineer	16	22.5
Team Member	5	7.0
Administrative Support	2	2.8
Other	4	5.6

As shown in Table 2, participants in the survey represented a variety of leadership levels and functions for software development. The participants represent the job functions for software development teams. Figure 1 provides a graphical representation of the demographic data for participant job function.

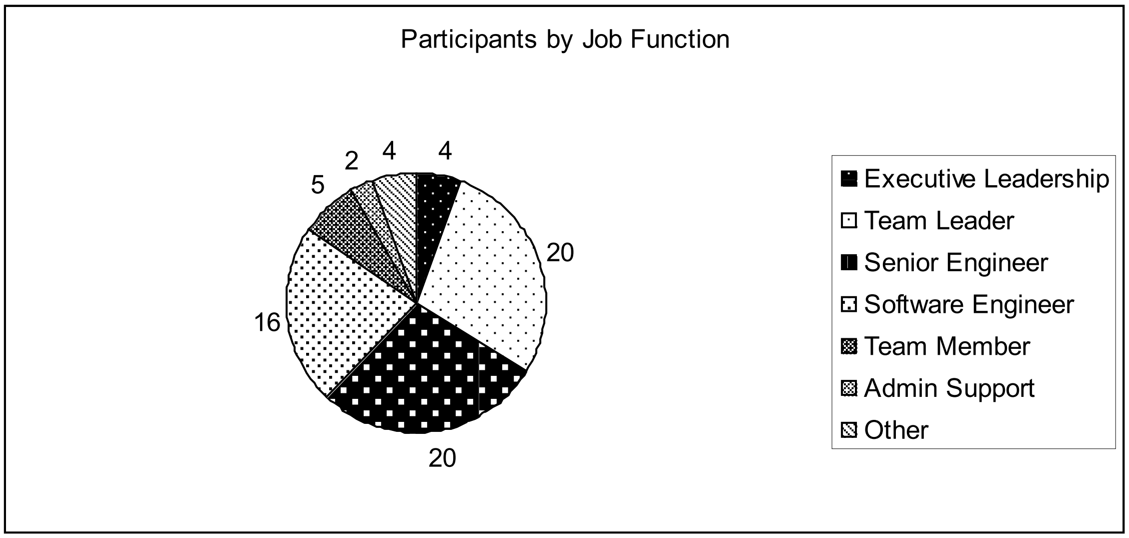


Figure 1. Frequency of participants by job function.

Table 3 provides and overview of the participant years of experience leading software engineering efforts.

Table 3

Participant Years of Experience Leading Software Development

Experience Leading Software Development	Frequency	Percentage
Less Than 1 Year	12	16.9
1 – 5 Years	18	25.4
6 – 10 Years	16	22.5
11 – 15 Years	10	14.1
16 – 20 Years	9	12.7
More Than 20 Years	6	8.5

As shown in Table 3, the experience level for leading software development programs ranged from less than 1 year to more than 20 years. The various leadership experience levels provided observations and views on software leadership from multiple perspectives. Figure 2 provides a graphical representation of the demographic data for years of experience leading software development efforts.

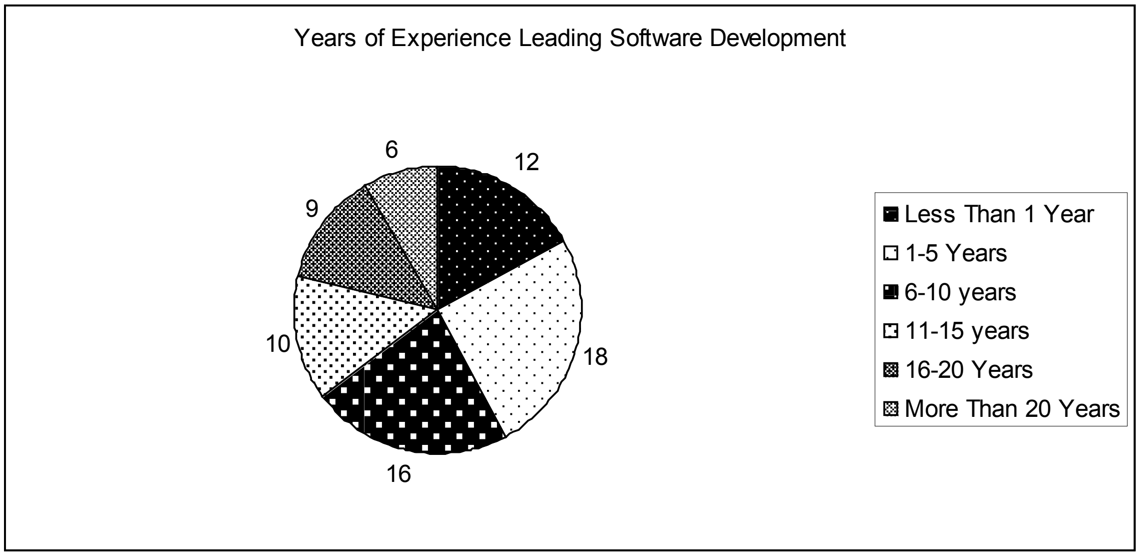


Figure 2. Participant years of experience leading software development efforts.

Table 4 provides and overview of the participant years of software engineering experience.

Table 4

Participant Years of Experience in Software Engineering

Software Engineering Experience	Frequency	Percentage
Less Than 1 Year	8	11.3
1 – 5 Years	13	18.3
6 – 10 Years	14	19.7
11 – 15 Years	8	11.3
16 – 20 Years	11	15.5
More Than 20 Years	17	23.9

As shown in Table 4, the years of software engineering experience range from less than 1 year to more than 20 years. The varied experience levels provided insight on software engineering processes and procedures from multiple perspectives. Figure 3 provides a graphical representation of the demographic data for years of experience in software engineering.

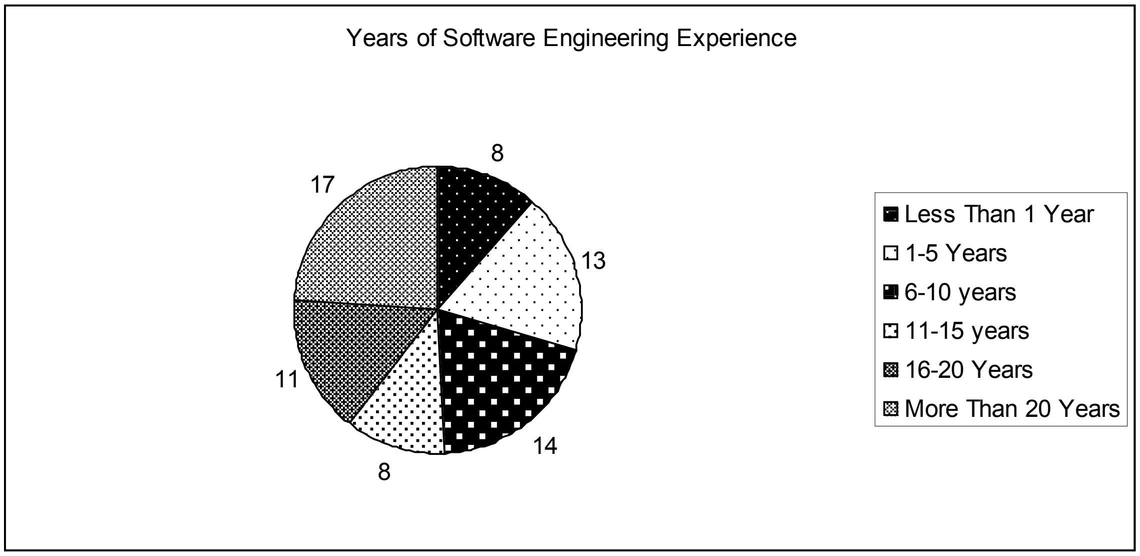


Figure 3. Participant years of experience in software engineering.

Data Collection Results versus Plan

The planned data collection approach for the research study distributed the electronic interview invitation to the e-mail distribution list containing 600 potential participants. The planned response rate of 5% projected 30 participants. The study invitation received an initial response from 158 individuals. After removing participants who did not agree to the informed consent, did not meet the sampling criteria, or failed to complete any of the open-ended interview questions, 71 participants contributed to the study. This resulted in a response rate of 12% versus the projected 5%.

The data gathering plan included 30 days for collection of data from respondents. The study invitation and link to the electronic interview questionnaire were available for 30 days with periodic reminders provided to the e-mail distribution list. Potential participants connected to the electronic questionnaire during the first 20 days of the data collection period. After 20 days, no additional participants connected to the questionnaire. The survey remained open for the 30 days as planned but did not result in additional participants.

Biases

The participant demographics for the study resulted in potential biases for the study results. Of the respondents completing the interview, 56.4% represented team leaders or senior system/software engineers. This percentage of senior personnel generates a potential bias in the results based on leadership perspectives and limited entry level or team member perspectives.

The selected research and development organization has achieved a CMMI Level 4 rating. As a result, the requirement to meet CMMI standards presents a bias in the participant responses for software development process and approaches. The organizational requirement for the use of CMMI procedures and standards represents a bias for participant responses.

Research Questions

This qualitative grounded theory research study focused on leadership practices for software engineering programs. The primary research question focused on the investigation of leadership impacts for program development. Research question 1: What leadership characteristics contribute to the positive outcome of software development programs? The electronic interview questionnaire presented a total of 11 questions. The first three questions obtained demographic information from participants. The remaining open-ended questions focused on leadership for software development. Questions 4, 5, 6, and 7 focused on the primary research question and sought to obtain views and perceptions on successful leadership characteristics and approaches for software development.

Interview Question 4 on Effective Leadership Approaches: What do you feel are the most effective leadership approaches for successful programs? Why do you feel these approaches are effective?

Interview Question 5 on Important Leadership Characteristics: What do you feel is the most important leadership characteristic to support software development programs? Why do you feel this characteristic is the most important?

Interview Question 6 on Successful Leadership Capabilities: In your experience, what leadership capabilities or approaches have contributed to the success of software development programs? Why do you feel they contributed to program success?

Interview Question 7 on Unsuccessful Leadership Approaches: In your experience, what leadership capabilities or approaches have contributed to the delay or failure of software development programs? Why do you feel they contributed to program delay or failure?

The supplemental research questions focused on software practices. Research question 2: What software practices contribute to the development of successful projects? Research question 3: What software practices result in negative impacts to projects?

Questions 8, 9, and 10 focused on the supplemental research questions and sought to obtain views and perceptions on software practices and methodologies.

Interview Question 8 on Successful Software Processes: What do you feel is the most important process for successful software development? Why do you feel this is the most important process?

Interview Question 9 on Effective Development Methodologies: What software development methodologies have you applied that are the most effective? Why do you feel these methodologies were the most effective?

Interview Question 10 on Ineffective Development Methodologies: What software development methodologies have you applied that are ineffective? Why do you feel these methodologies were ineffective?

Question 11 of the electronic interview questionnaire provided the participants with the opportunity to provide any additional insights, thoughts, or information on leadership and software engineering.

Interview Question 11 Additional Comments: Please provide any additional comments or insights you would like to share on software leadership capabilities or software processes.

Themes Identified

The themes identified were derived from data coding and analysis of the qualitative data collected through the electronic interview questionnaire. The data collected from the 71 respondents providing answers to one or more open-ended questions was coded and reviewed for common themes and patterns. Participation in the study was anonymous and each participant was assigned a number from 1 to 71 for coding. Data coding supported examination of the open-ended responses obtained to identify emerging themes and concepts.

Section 3 of the electronic interview questionnaire consisted of eight open-ended questions on software leadership, processes, and methodologies. The questions focused on obtaining participant views for leadership and software engineering. The open-ended responses were reviewed and coded to identify themes for successful software development and leadership.

The open-ended questions focused on software leadership, software processes, and software development methodologies. Table 5 provides an overview of the eight open-ended questions and the total responses for each question. The exact wording of each question is provided in the interview questionnaire (see Appendix D).

Table 5

Open-ended Questions and Respondents

Question Number and Topic	Number of Respondents
Question 4: Effective Leadership Approaches	68
Question 5: Important Leadership Characteristics	69
Question 6: Successful Leadership Capabilities	57
Question 7: Unsuccessful Leadership Approaches	61
Question 8: Successful Software Processes	54
Question 9: Effective Development Methodologies	50
Question 10: Ineffective Development Methodologies	46
Question 11: Additional Comments	37

For coding purposes, the participant number remained the same for each question in the survey. Appendix E provides the specific demographic information for the respondents to each open-ended question. Analysis and coding of responses provided by participants resulted in the identification of themes for software engineering leadership, software resources, and software processes.

Data Analysis

Qualitative research involves the review, organization, and analysis of collected data to determine the perception of participants (Shank, 2006). Data coding sorts the data into broad categories for further analysis (Neuman, 2003). Data coding supports identification of emerging themes and patterns in participant responses (Shank). In qualitative research, data coding is performed in several stages to manage and refine the large amount of textual data collected (Creswell, 2008; Neuman).

In this research study, the data collected from the 71 participants was reviewed for familiarity with data content and potential categories. Initial sorting of the data was conducted manually using index cards and whiteboards to identify potential categories for the data. A second review refined the data and developed core categories for analysis. After the initial data review and category identification, further analysis and categorization was performed utilizing qualitative data analysis software. The NVivo 8 (QSR International, 2008) qualitative data analysis software supported additional examination and classification of the textual information collected from participants.

The data analysis included review of the responses for common themes, words, and concepts. Alignment of common categories supported identification of themes for developing new leadership theory for software engineering. Analysis of identified themes in the data resulted in identification of 13 success drivers for software development and leadership. A second phase of analysis categorized the success drivers within the theoretical framework of the open system paradigm.

The 13 key success drivers or themes were categorized based on the open system paradigm. Open systems theory integrates resources, environment, and processes to develop a cohesive synergistic approach and structure to leadership (National Defense University, 2009). The open system paradigm incorporates the concept of continual process improvement through monitoring, modifying, and improving processes and procedures throughout the program life cycle (Bloch, 2008). Leadership in the open system paradigm improves performance through integration of resources, environment, and processes to meet defined goals and objectives (Lewis, 2006). The open system paradigm provides the framework for improved leadership through the development of integrated approaches for success.

The data analysis process reviewed the 13 identified themes and identified three categories related to open system theory: environment, resource, and process. The themes that emerged from the data related to leadership approaches, procedures, software development techniques, methodologies, and processes. Relating the themes to the open system paradigm resulted in the identification of the three key categories of leadership environment, implemented resources, and software processes. Each of these categories provides a critical component required for software engineering success.

The leadership environment category identifies the key themes related to development of a successful leadership environment. The software resource category focuses on identification of the themes that provide critical software engineering resources contributing to successful programs. The software process category identifies the software development methods, procedures, and process that result in positive program performance. The integration of these three themes becomes the framework for grounded theory development on successful software engineering leadership.

The data analysis identified 13 key themes for software development success. The analysis of these themes revealed the three categories of environment, resources, and processes for successful leadership. Results of the data analysis supported theory development on leadership environment, software resources, and software process for successful software engineering efforts.

Coding the Data

According to Creswell (2008), "coding is the process of segmenting and labeling text to form descriptions and broad themes in the data" (p. 251). Coding data involves reviewing participant input and identifying codes to combine to create resulting themes. The process of coding the data in this study followed basic steps as defined by Creswell of: (a) reviewing initial data, (b) identify segments of information, (c) code the segments, (d) combine codes to reduce redundancy, and (e) identify common themes from codes.

Review of collected data provided an overview of the concepts and possible codes. The individual questions represented segments of information to provide an orderly process for review and coding. Review of responses provided an initial set of codes for further review. Combining overlapping and redundant codes resulted in a manageable set of codes for analysis. The final set of codes resulted in identification of themes from participant responses.

Coding for Software Engineering Leadership. Analysis and coding of the respondent data resulted in the identification of themes related to software engineering leadership. The coding process identified initial codes, developed common codes, and derived themes. Figure 4 provides a representation of the coding process, sample codes identified, and resulting themes derived related to software engineering leadership.

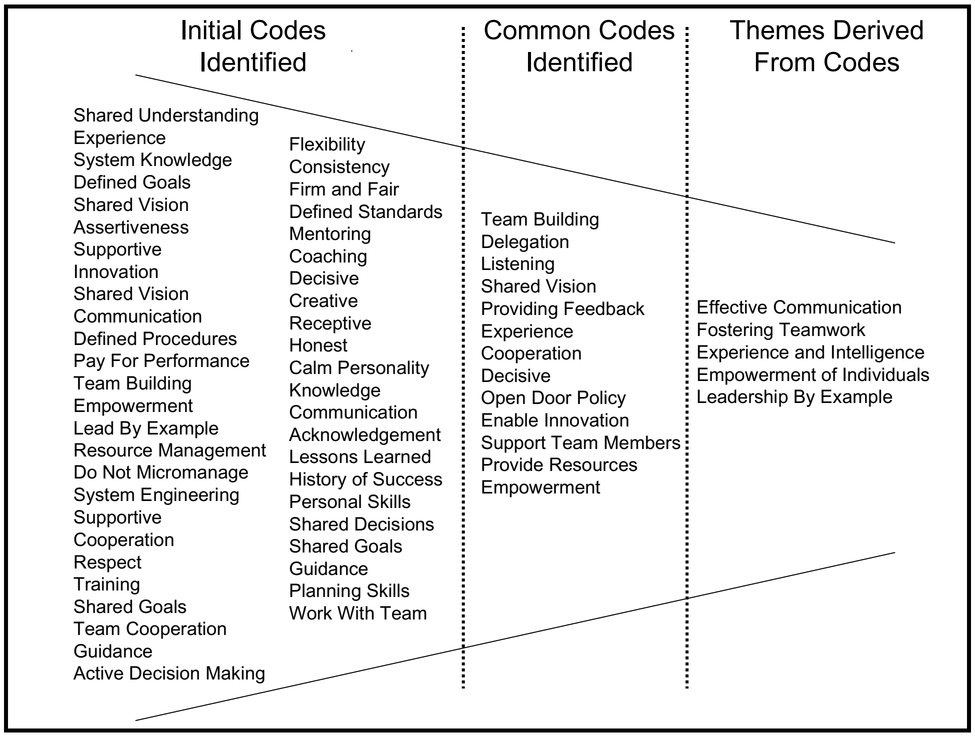


Figure 4. Coding process, sample codes, and derived themes for software engineering leadership.

Coding for Software Engineering Resources. Analysis and coding of the respondent data resulted in the identification of themes related to required software engineering resources. The coding process identified initial codes, developed common codes, and derived themes. Figure 5 provides a representation of the coding process, sample codes identified, and resulting themes derived related to software engineering resources.

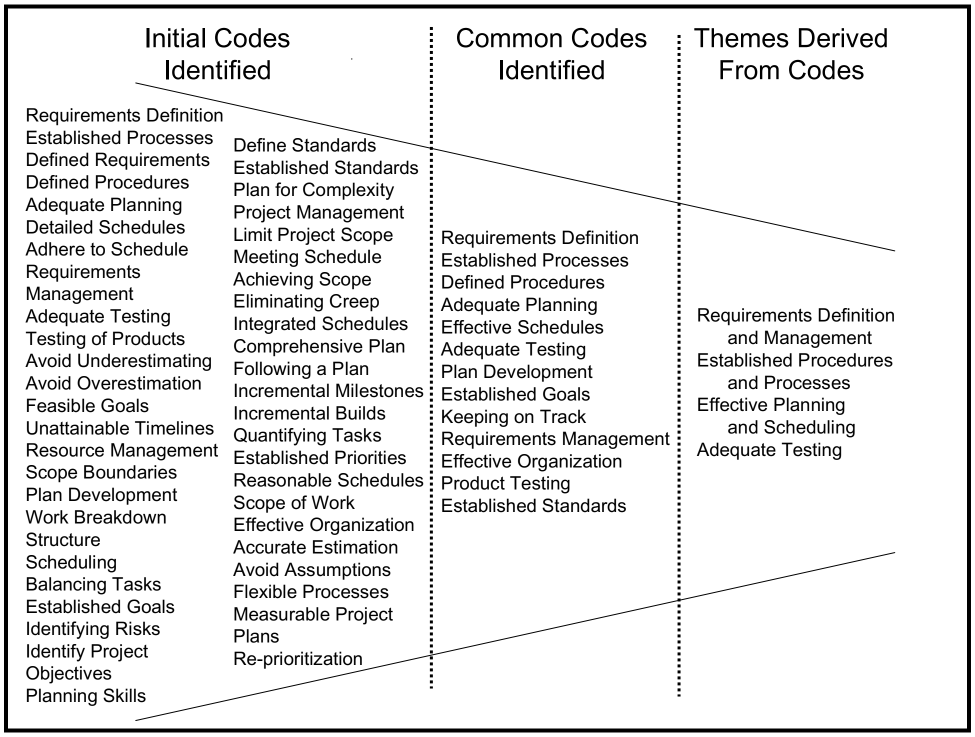


Figure 5. Coding process, sample codes, and derived themes for software engineering resources.

Coding for Software Engineering Processes. Analysis and coding of the respondent data resulted in the identification of themes related to software engineering processes. The coding process identified initial codes, developed common codes, and derived themes. Figure 6 provides a representation of the coding process, sample codes identified, and resulting themes derived related to software engineering processes.

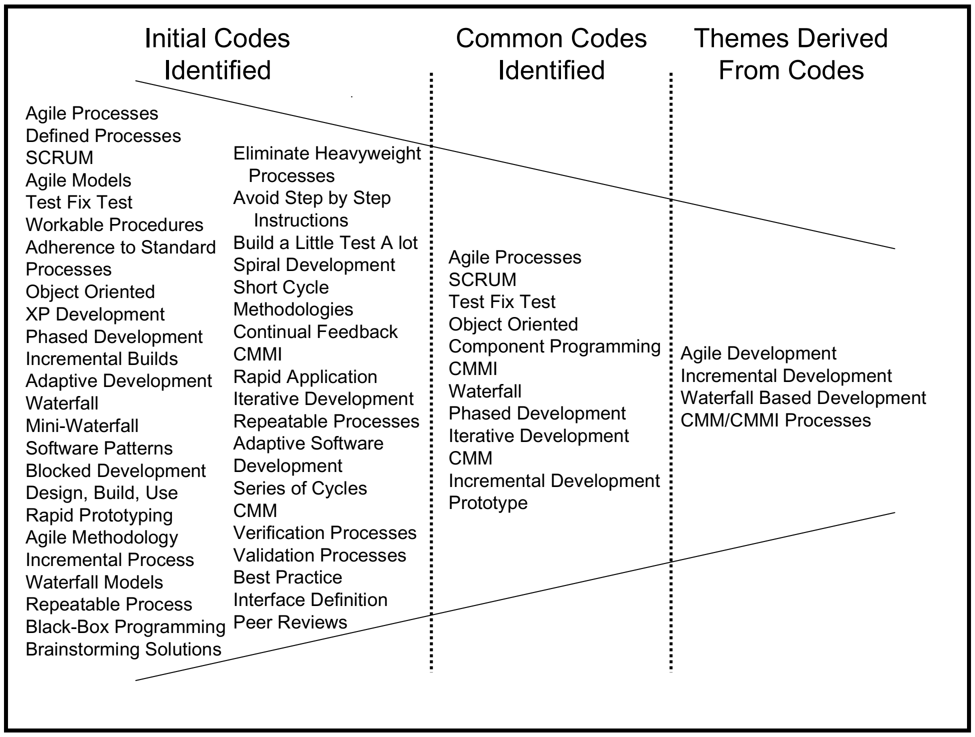


Figure 6. Coding process, sample codes, and derived themes for software engineering processes.

Question 4: Effective Leadership Approaches

Question 4 focused on effective leadership approaches. Participants were asked what they thought the most effective leadership approaches were for successful software development programs. The 68 respondents provided data that resulted in the three themes for effective leadership approaches of empowerment of individuals, fostering teamwork, and effective communication. Frequency of themes reported in responses is provided in Table 6. The column labeled participants indicates the number of individuals providing a response related to the theme. The column labeled frequency indicates the number of occurrences in the data related to the theme. Example responses for each theme are provided below.

Table 6

Themes for Question 4 Effective Leadership Approaches

Theme	Participants	Frequency
Empowerment of Individuals	24	47
Fostering Teamwork	27	48
Effective Communication	18	24

Theme 1: Empowerment of Individuals

Participant 1: "Employ people you can trust and then entrust them to do their job. You don't have to know it all, you just need to know how to get the know-it-all to perform and report back to you."

Participant 9: "Empowerment and responsibility shared among all members of the team. Give ownership and identity to the IPTs so they can truly appreciate their role in developing a quality product."

Participant 14: "Allowing a team to do their job without micromanagement. This approach is effective because it allows the team to express their ideas freely and possibly present new ideas to the project."

Participant 45: "Choose qualified people for the job, make sure they understand the task, and get out of their way. Let them do their work."

Participant 59: "Emulate any successful sports coach. Recruit members that are talented and motivated. Call the next few plays to get the game started. Stand back and watch them run."

Theme 2: Fostering Teamwork

Participant 3: "Teamwork and mentoring approaches so the developers can build based upon more than just a specifications sheet."

Participant 25: "My focus in my leadership activities is the motto: None of us are as knowledgeable as all of us. This is to build teamwork and help everyone on the team to know that they have valuable ideas to contribute."

Participant 48: "Teamwork. The type of teamwork where everyone is respected for what they do or can/cannot do. When a team feels connected nothing will stop them in achieving the team goals."

Theme 3: Effective Communication

Participant 8: "A leader who communicates well with their team can establish an effective open-door policy. This will allow the lead to identify risks early, manage risk, and accurately assess how the project is doing."

Participant 12: "I feel that an open leadership is necessary for successful programs. Everyone needs to be able to discuss with each other problems and successes to be able to understand what each team member might need from other team members."

Participant 34: "Communication, goals of the organization should be communicated throughout the organization."

Question 5: Important Leadership Characteristics

Question 5 focused on identification of important leadership characteristics for successful development programs. Participants were asked what they thought the most important leadership characteristics were for successful software development programs. The 69 respondents provided data that resulted in the four themes of effective communications, leadership by example, experience and intelligence, and empowerment of individuals. Frequency of themes reported in responses is provided in Table 7. Example responses for each theme are provided below.

Table 7

Themes for Question 5 Important Leadership Characteristics

Theme	Participants	Frequency
Effective Communication	31	51
Leadership by Example	22	37
Experience and Intelligence	18	28
Empowerment of Individuals	10	15

Theme 1: Effective Communication

Participant 10: “Communication to assure common understanding.”

Participant 24: “Clear and concise communication – whether communicating verbally, via pencil and paper, or electronically, ensure that all parties have a clear and concise understanding of the exchange.”

Participant 34: “Communication, goals of the organization should be communicated throughout the organization.”

Participant 37: “Communication. If leaders cannot successfully communicate the needs and status how can team members have any hope of producing what is needed. If team members cannot successfully communicate their needs, issues, and successes how can their lead know the accurate status of the project and guide tem to a successful delivery.”

Theme 2: Leadership by Example

Participant 5: "Lead by example and always stick to your word. Honesty will get the best out of those under the leader."

Participant 31: "A leader must be willing to do what they ask others to do. An employee will be loyal to a manger who shows the same loyalty back. This not only includes compliments for a good job but constructive criticism in areas where improvement is needed with possible solutions to the issue and help achieving them."

Participant 52: "Demonstrate leadership support. Lead by example."

Theme 3: Experience and Intelligence

Participant 23: "Ensuring that all involved are trained to the highest degree possible so the results can not be questioned."

Participant 28: "First: Technical knowledge and comprehension. A leader that cannot comprehend or relate to the underlying science, technology, or domain loses a lot of respect from those he is leading."

Participant 44: "It is a qualification knowledgeable about system engineering and internal organizational standard teamwork. This type of knowledge promotes and empowers the team to support the software development programs to be successful."

Theme 4: Empowerment of Individuals

Participant 3: "Guidance as opposed to dictatorial specifications. This allows the developer a chance to use innovative approaches."

Participant 25: "Hire talented personnel and trust them to do their work."

Participant 27: "Realizing that you do not have all of the answers and depending on your team to provide the ones you don't know. Treating your team with respect."

Participant 69: “The ability to trust your team; to turn loose the reigns and let them make decisions, so that they have a sense that the software product belongs to them. With project/product ownership the team will put their best efforts forward in building the system and take pride in what they have created together as a team.”

Question 6: Successful Leadership Capabilities

Question 6 focused on identification of leadership capabilities resulting in successful software development programs. Participants were asked what leadership capabilities they thought resulted in successful software development programs. The 57 respondents provided data that resulted in the four themes of effective communications, fostering teamwork, experience and intelligence, and empowerment of individuals. Frequency of themes reported in responses is provided in Table 8. Example responses for each theme are provided below.

Table 8

Themes for Question 6 Successful Leadership Capabilities

Theme	Participants	Frequency
Effective Communication	29	54
Fostering Teamwork	18	25
Experience and Intelligence	17	29
Empowerment of Individuals	7	15

Theme 1: Effective Communication

Participant 9: “Leaders that continuously communicate with the development teams and provided the big picture view, by use of integrated and detailed schedules or by other means. Trusted and supported the decision, analysis, and trade-offs proposed by the development team (with appropriate back-up data)."

Participant 17: "Communication: This is key to open all available information to everyone for a well rounded research of the issue."

Participant 24: "Establishing open lines of communications – fostering an environment of open communication both formally and informally will help ensure the right questions are being asked and the issues are being brought forward."

Theme 2: Fostering Teamwork

Participant 8: "A team-centric, honest approach will allow most efforts to be completed successfully."

Participant 19: "Team cohesiveness is very important to the success of a SW development project. A team that works well together can almost run on autopilot, however on the other hand, a mismatched team requires constant management attention."

Participant 35: "Maximizing team efficiency through specialization is, in my experience, the best path to program success."

Theme 3: Experience and Intelligence

Participant 30: "Technical detail oriented leadership. A person in charge who understands the technical approach and its potential challenges/issues will be better prepared."

Participant 32: "Technical competence, work ethics, concern for the project and for the people working the project."

Participant 57: "Having a boss/leader that has technical knowledge as well as good management skills is helpful because I can ask technical questions when I need help on a particular assignment while he/she can also help the program advance by broadening our customer base with good management skills."

Theme 4: Empowerment of Individuals

Participant 19: "It is also very important to not only delegate responsibility for tasks, but also the authority to do what is necessary to complete assignment tasks. A manager who is not efficient at delegating will quickly become overwhelmed."

Participant 46: "The ability to guide without being overbearing. This will help with employees being more willing to be productive in order to make their boss look good."

Participant 69: "A management hands off approach (no mandates), letting the team brainstorm their own ideas, coming to consensus, and developing a sense of ownership in the product and process used to make the product."

Question 7: Unsuccessful Leadership Approaches

Question 7 focused on identification of leadership approaches that contributed to unsuccessful development programs. Participants were asked what leadership approaches they thought resulted in unsuccessful software development programs. The 61 respondents provided data that resulted in the five themes of indecisive or inadequate leadership, inadequate planning, inadequate communication, inadequate experience or knowledge, and arrogant or ego driven leadership. Frequency of themes reported in responses is provided in Table 9. Example responses for each theme are provided below.

Table 9

Themes for Question 7 Unsuccessful Leadership Approaches

Theme	Participants	Frequency
Indecisive or Inadequate Leadership	21	36
Inadequate Planning	20	40
Inadequate Communication	14	27
Inadequate Experience or Knowledge	14	22
Arrogant or Ego Driven Leadership	8	14

Theme 1: Indecisive or Inadequate Leadership

Participant 1: “Making a hasty decision or a decision without hearing all of the necessary inputs. This causes many problems in the future.”

Participant 11: “Leadership that can’t make decisions with input from team members and other stakeholders. Inability to work on a team (emotional intelligence). Inadequate technical knowledge can sometimes be the cause of the indecisiveness.”

Participant 43: “The ones that sit in the ivory tower, you should bow down to me and refuse to take input. Ones that waffle and give the customer everything with no consideration to cost or schedule.”

Participant 69: “Ineffective management when decisions aren’t made at all, teams can become like a rudderless ship adrift in a sea of work. Forcing a team to do more than their resources will allow always dooms a development from the start.”

Theme 2: Inadequate Planning

Participant 2: “Allowing scope creep. Schedule gets destroyed.”

Participant 9: “Leadership that focuses on daily fire fighting without the proper focus of a future vision and plan. Urgent tasks prioritized, but they may not be the important tasks. Lack of leadership attention to requirements definition and system modeling.”

Participant 21: “Inadequate planning and correctly quantifying tasks usually causes team to fail to identify the risks and needs of a project.”

Participant 46: “Unattainable timelines. Some managers push to meet or exceed goals that are not feasible depending on the type of project.”

Theme 3: Inadequate Communication

Participant 10: “Poor communications, lack or planning (or sharing the plan) avoiding decisions until default solution is reached.”

Participant 16: “Usually delays or failures result from an inability to effectively communicate. Assumptions about successful communication early in the development cycle that are shown to be inaccurate later result in great frustration, compounding the issue of getting these assumptions corrected later in the cycle. Fixing the issues combined with a team that still is unable to effectively communicate makes for a very difficult task.”

Participant 37: “Failing to communicate all information available. Holding back details because the team lead felt the team did not need to know. The team leader failing to know the difference. The team had false assumptions about what they were supposed to do and when.”

Participant 71: “Communication. Communication is at its best when it is bi-directional and at its lowest when the communication traffic is one-way.”

Theme 4: Inadequate Experience or Knowledge

Participant 12: “Lack of knowledge of the process can contribute to the delay or failure of a program. If do not understand the process or even why there is a process then you overlook many things that need to be done and they do not get scheduled resulting in a late project and usually over budget.”

Participant 57: “It has delayed SW development when team leaders do not understand the technical importance or place of a particular assignment or component. This inability to understand the technical importance renders him in able to direct teammates properly or to reference teammates where to find proper help when needed.”

Participant 64: “One not being able to develop quickly enough to deliver before a program need or technology passes you by.”

Theme 5: Arrogant or Ego Driven Leadership

Participant 20: “Tyrannical/autocratic leaders typically alienate the team, leading to a loss of morale and productivity. Another problem I’ve seen is the manger that’s more worried about looking good to the customer/upper management and not taking care of the folks below them. Again, this ruins morale and drops productivity.”

Participant 23: “Believing you are smarter than the requirements and that you can do what you want, even though the requirements say different.”

Participant 48: “Arrogance in oneself or one’s buddies can delay or cause a program to fail; others are turned off by arrogance and actually wish for failure.”

Participant 50: “Stubbornness, thinking you know all.”

Question 8: Successful Software Processes

Question 8 focused on identification of software processes for successful development programs. Participants were asked what software processes they thought resulted in successful software development programs. The 54 respondents provided data that resulted in the four themes for successful development of requirement definition and management, establishing procedures and processes, effective planning and scheduling, and adequate testing. Frequency of themes reported in responses is provided in Table 10. Example responses for each theme are provided below.

Table 10

Themes for Question 8 Successful Software Processes

Theme	Participants	Frequency
Requirement Definition and Management	21	48
Established Procedures and Processes	18	42
Effective Planning and Scheduling	11	26
Adequate Testing	5	10

Theme 1: Requirement Definition and Management

Participant 6: “Clearly defining the requirements. Cannot obtain what you cannot define.”

Participant 11: “Thorough requirements development and management. Including effort estimates with adequate rationale and assumptions. This provides the road-map, keeps the team on track (avoid scope-creep), and as issues arise with a particular requirement gives the management team a process to drop that requirement or estimate how changes will impact cost and schedule.”

Participant 30: “Successfully understanding the stakeholders’ requirements is most important since it affects the rest of the life cycle. Rework can be very costly if requirements are misunderstood.”

Participant 53: “Requirements definition! Think about them, discuss them, identify them, and freeze them. A moving target is not a recipe for success.”

Theme 2: Established Procedures and Processes

Participant 1: “Establish a process and plan up front and stick to it, but be flexible enough to adjust the processes and plans as necessary. Must do this wisely so that changes are not made in haste to fix a single issue, but can be applied to the entire program.”

Participant 44: “It is a defined internal organization process. Defined internal organization process successfully provides manageable software program development within the program timeframes and budgets.”

Participant 52: “Structured/repeatable SW development processes. Anyone can come into a project and contribute without completely understanding the hardware system they are supporting.”

Participant 70: “I can’t really name one process, but a development process that has been fully documented and proven to work via validation/verification will work. Every developer has their own process. Some are good, and some are bad. But the ones, who document their development process and can show you where they are in the process at any time you ask should work.”

Theme 3: Effective Planning and Scheduling

Participant 25: “PLANNING, Planning, and Planning.”

Participant 37: “Communication, planning, and ability to adapt. Planning must take into account roadblocks, skill sets of team, risk factors, yet include a dose of reality.”

Participant 46: “The planning stage. If errors can be caught in the planning stage, the cost is lower than it would be to find errors in the test and analysis stage.”

Participant 68: “Planning, without planning you have no foundation.”

Theme 4: Adequate Testing

Participant 31: “Testing. If a customer gets a buggy piece of software presented as a finished product, the whole team looks bad.”

Participant 42: “Functional testing. Flaws must be found before software ships.”

Participant 59: “Requirements. Requirements. Requirements. Followed by test, test, and more test.”

Question 9: Effective Development Methodologies

Question 9 focused on identification of software development methodologies for successful development programs. Participants were asked what software development methodologies they thought resulted in successful software development programs. The 50 respondents provided data that resulted in the four themes of iterative development, agile development, waterfall development, and CMM/CMMI. Frequency of themes reported in responses is provided in Table 11. Example responses for each theme are provided below.

Table 11

Themes for Question 9 Effective Development Methodologies

Theme	Participants	Frequency
Iterative Development	11	18
Agile Development	11	21
Waterfall Development	6	6
CMM/CMMI	5	6

Theme 1: Iterative Development

Participant 9: "Incremental build approach. Easily defined and measured for program status, and can be defined based on program needs and risks."

Participant 29: "The most effective methodologies I have used would be XP or spiral model development methodologies. These methodologies take changes into consideration while in development. There is a communication gap between software developers and stakeholders. The more contact with the customer assures that at some point this gap is bridged together."

Participant 33: "Iterative development. Taking customer feedback to refine future requirements is critical to creating a helpful product."

Participant 66: "Iterative and incremental development."

Theme 2: Agile Development

Participant 7: "Agile development."

Participant 15: "Adaptive software development. I have been using agile development for the last 10 years and it always delivers something that works. It may not be exactly what we wanted in the beginning but it works and meets the customer's expectations."

Participant 20: "Rapid prototyping or what's now being called SCRUM. Doing the development as a series of small development cycles. This allows you to verify that you've understood the customers desires much quicker and allows you to minimize any rework required if it turns out you didn't."

Participant 32: "Agile or SCRUM. At the end of each sprint, stakeholders and team members can meet to assess the progress of a project and plan its next steps. This allows a project's direction to be adjusted based on completed work not speculation or predictions."

Theme 3: Waterfall Development

Participant 28: "It depends on the nature of the project and the organization, but honestly, a traditional waterfall methodology is often very effective when use appropriately, when the domain is well understood and the requirements can be sufficiently specified early on."

Participant 53: "Waterfall using an effective verification and validation have worked best for me. If the V&V is performed properly it will establish traceability for all requirements to be fulfilled."

Theme 4: CMM/CMMI

Participant 6: "CMM. Focus on the basics."

Participant 27: "CMMI is very effective as long as it is within reason. Process for the sake of process is not good. Process that aids in implementing proper solutions is invaluable."

Participant 71: “CMMI because you gain an opportunity to investigate the software methodologies that are being used or have a methodology criteria to base ones against.”

Question 10: Ineffective Development Methodologies

Question 10 focused on identification of software development methodologies that resulted in unsuccessful development programs. Participants were asked what software development methodologies resulted in unsuccessful software development programs. The 46 respondents provided data that resulted in the two themes of lack of management not process and waterfall development. Frequency of themes reported in responses is provided in Table 12. Example responses for each theme are provided below.

Table 12

Themes for Question 10 Ineffective Development Methodologies

Theme	Participants	Frequency
Lack of Management not Process	17	17
Waterfall Development	8	8

Theme 1: Lack of Management Not Process

Participant 6: “Most. Gimmicks that focus on ineffective details of engineering and not on lack of management/leadership.”

Participant 9: “It has been my experience that it is not the methodology that was at fault or ineffective, but instead the poor implementation of any methodology.”

Participant 69: “Poor planning and asking the team to do more that was possible given the resources the team had to work with.”

Theme 2: Waterfall Development

Participant 7: “Stringent, heavyweight processes such as Waterfall methods, which are very management driven tend to stifle the sense of ownership and self-management that are critical, particularly to the younger generation of developers.”

Participant 15: “Waterfall does not work in development projects.”

Participant 29: “Any type of waterfall methodology just ends badly.”

Participant 67: “Waterfall – long cycles and dynamic personnel situations make this method long in the tooth.”

Question 11: Additional Comments

Question 11 allowed participants to provide any additional comments, observations, or views on leadership, software development, and processes. Participants were asked to provide any additional comment as desired. The 37 respondents provided data that emphasized the themes identified in the previous questions. Participants stressed the importance of communication, teamwork, empowerment, and experience for effective leadership. Participants noted good leadership skills are essential to successful software development. Example responses are provided below.

Participant 1: “A good leader can lead anything. They don’t have to be a great software leader to successfully lead a software program…just a good leader with wise decision making capabilities.”

Participant 29: “To be a successful lead, there is a lot more than knowing many types of processes or methodologies. It is almost more important to understand you counter-parts and understand how they approach processes.”

Participant 57: “It is important to keep everyone informed on all sides of software development.”

Participant 69: “Good team leaders communication with their folks, empower them to make decisions and suggest changes to the development process.”

Summary

This qualitative grounded theory study investigated leadership and software development practices applied to software development programs. The goal of the study was to explore and analyze leadership and development processes resulting in successful software development programs by obtaining views and perceptions of software developers, leaders, and team members. The research explored the experiences and unique perceptions of leaders and software developers actively involved in software development programs.

Data for the study was collected using an Internet questionnaire to obtain views, perceptions, and beliefs from software engineering experts and team members. The electronic questionnaire contained four sections; assurance of confidentiality and informed consent, closed-ended demographic questions, open-ended questions on software development, and a final opportunity to withdraw from the study without submitting responses. The study responses were analyzed to identify trends, themes, characteristics, and behaviors supporting successful software programs. Data coding sorted the data into broad categories and data sorting identified emerging themes and patterns related to software development and leadership.

Data was collected for analysis from the respondents agreeing to the informed consent, providing answers to the demographic questions, meeting the sampling criteria, providing answers to one or more of the open-ended questions, and electing to submit answers in section 4 of the questionnaire. Data collection resulted in 71 participants providing views and perceptions on leadership and software engineering.

Data coding and analysis resulted in identification of themes for software engineering leadership and software development processes. For successful software leadership the five themes of empowerment of individuals, fostering teamwork, effective communication, experience and intelligence, and leadership by example were identified. Data collected on unsuccessful leadership characteristics and approaches resulted in the five themes of indecisive or inadequate leadership, inadequate planning, inadequate communication, inadequate experience or knowledge, and arrogant or ego driven leadership.

Secondary research questions focused on identification of effective processes and methodologies for software development. For successful software processes the four themes of requirement definition and management, established procedures and processes, effective planning and scheduling, and adequate testing were identified. Data analysis identified the four themes for effective development methodologies of iterative development, agile development, waterfall development, and CMM/CMMI approaches.

Chapter 4 presented the findings of the qualitative grounded theory study on software engineering leadership. Chapter 5 provides an analysis of the identified themes and theories developed. Chapter 5 presents conclusions, implications, and recommendations on software engineering leadership and software process application that emerged from the study.

CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

Chapter 4 presented the results of the data collection for this qualitative grounded theory study. The textual data collected was reviewed for themes related to software engineering leadership. The electronic questionnaire provided views on software engineering leadership, software development, software processes, and software methodologies. Analysis of collected data produced several themes related to software development and leadership. The collected data and identified themes were presented without interpretation of the data or development of theory.

The purpose of the study was to investigate leadership and development practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. The qualitative study implemented a grounded theory research methodology for data collection and analysis. Interpretation of the data contributed to the development of new software engineering leadership theory from the identified themes. Chapter 5 provides an interpretation of the data and themes as a result of the grounded theory analysis.

Conclusions

This qualitative grounded theory research study investigated the views and perceptions of software engineering leaders and team members. The electronic questionnaire obtained views and perceptions on leadership practices for software engineering programs. The primary research question investigated leadership impacts for program development. The supplemental research questions investigated successful software practices. The research questions guided the study to investigate and identify processes, practices, and leadership approaches that contribute to successful software development programs.

Interpretation of the research findings contributed to the development of new leadership theory for successful software development programs. The data collected and themes identified support theory development through the theoretical framework of open systems. The research questions framed the study focusing on software leadership and software practices. Interpretation of the findings resulted in identification of new theory for successful software engineering leadership.

Research Questions and Theory Development

The primary research question investigated leadership practices for software engineering programs. Research question 1: What leadership characteristics contribute to the positive outcome of software development programs? The supplemental research questions investigated software practices. Research question 2: What software practices contribute to the development of successful projects? Research question 3: What software practices result in negative impacts to projects?

Analysis of the collected data resulted in the identification of 13 critical themes for successful software development. These 13 themes resulted in the identification of the three categories of leadership environment, software resources, and software process. These themes and categories formed the basis of the development of a leadership theory for software development.

Leadership Environment Category

Open-ended questions 4, 5, 6, and 7 of the electronic interview questionnaire investigated the primary research question and sought to obtain views and perceptions on successful leadership characteristics and approaches for software development programs. The interview questions investigated leadership approaches, capabilities, and characteristics perceived to support successful software development initiatives. Analysis of the data revealed five common themes for successful leadership: (a) effective communications, (b) fostering teamwork, (c) experience and intelligence, (d) empowerment of individuals, and (e) leadership by example. Table 13 provides a summary of the frequency for each of the themes in the participant responses.

Table 13

Frequency of Environmental Themes for Participant Responses

Theme	Frequency
Effective Communications	129
Fostering Teamwork	73
Experience and Intelligence	63
Empowerment of Individuals	77
Leadership by Example	51

These five themes relate to the development of a leadership environment contributing to successful software development programs. Each of these themes provides a critical component for leadership environment. Participants stressed the importance of these components for leadership in achieving successful outcomes.

Effective communications. Participants stressed effective communications throughout the responses. For the 71 participants, open-ended responses referenced communications 129 times in questions 4, 5, and 6. Question 7 asked for perceptions on ineffective leadership approaches. The participants sited ineffective communications as the reason for program failure 27 times in the responses. This result emphasizes the importance of effective communications in supporting successful software development environments.

Mohtashami et al. (2006) observed effective communications are essential to successful software development teams. Effective communication of objectives and open communication throughout development by leadership fosters improved collaboration (Bharadwaj & Saxena, 2006) and provides the framework for leadership success (Nielsen, 2009). For successful software development, leaders must stimulate creativity, enable achievement of objectives, and articulate a clear vision through effective communications (Denning, 2007; Nielsen).

Effective communications contributes to program success while ineffective communications prohibit success. Communications are not only essential to program success but also lack of communication or ineffective communication can be detrimental to program development. Participant views revealed communications is a critical factor to achieving leadership and program success in the software engineering environment.

Fostering teamwork. Participants mentioned teamwork 73 times in questions 4 and 6 on effective leadership approaches and leadership capabilities for success. Team development, support, and participation are perceived as essential to leadership success. However, in question 7 lack of teamwork or team development was only cited twice as a factor for ineffective leadership. Enabling teamwork, supporting team objectives, and contributing to team activities contribute to successful leadership of software programs.

Fostering teamwork has a positive effect on software development leadership but lack of teamwork is not a primary contributor to negative program outcomes.

Software engineering programs achieve objectives through the integration of experts, information, and knowledge (Neumann, 2008; Peppard et al., 2007). Innovations in technology and networking capabilities demand complex, integrated, and innovative software systems to meet emerging customer requirements (Hadar & Leron, 2008; Kirova et al., 2008; Schneidewind, 2007). Successful development of complex software systems requires collaborative software teams developing software applications meeting consumer quality, reliability, and functionality demands (Brown & McDermid, 2008; Lee et al., 2006). Developing an innovative and creative environment fostering teamwork contributes to the achievement of critical software development objectives.

Experience and intelligence. Experience was present in responses to questions 4, 5, and 6. Participants referenced the importance of experience, knowledge, and intelligence for leadership in 63 instances. In question 7, participants cited inadequate knowledge or experience 22 times as a factor in ineffective leadership. Knowledge, intelligence, and experience of leadership are critical to achieving program success. The lack of knowledge and intelligence contributes to program failures.

Knowledge is a critical asset requiring effective management to sustain competitive advantage (Mathew, 2008). In the software development environment, knowledge supports decision-making, problem solving, and achievement of objectives (Mathew & Kavitha, 2008). The complex software development life cycle requires knowledge of processes, methodologies, practices, and approaches for success (Landaeta, 2008).

Participants in the study viewed knowledge and intelligence as critical attributes for effective leadership. Leadership without knowledge or experience resulted in negative impacts to program success. Experience, knowledge, and intelligence are integral to the success and effectiveness of leadership for software development programs.

Empowerment of individuals. Participants mentioned the importance of empowerment throughout the responses. In questions 4, 5, and 6 empowerment was referenced 77 times as an attribute contributing to successful program outcomes. In question 7, failure to empower individuals was not listed as a contributor to program failure. However, similar characteristics such as indecisive leadership, arrogant leadership, and ego driven leadership were cited as reasons for program failures.

Northouse (2010) observed the complex, integrated, and dynamic software development environment requires leaders to empower team members to meet goals and objectives. Empowered individuals accept ownership, take responsibility, exercise self-discipline, and increase efficiency (Cagle, 2007; Chan et al., 2008). Leaders can encourage innovation, creativity, productivity, and proficiency to flourish through development of environment empowering individuals (Cagle; Chan et al.).

In conjunction with fostering teamwork, empowerment of individuals is a critical component for software development program success. The complexity of software efforts requires integrated individuals empowered to meet objectives and program goals. Individuals and teams require empowerment for task completion, decision-making, and problem solving to achieve critical software program objectives.

Leadership by example. Participants referenced leadership by example 51 times in responses to questions 4, 5, and 6. In question 5, identification of the most important leadership characteristics for success, leadership by example was listed 37 times as critical to program success. Participants also noted the importance of removing ego, arrogance, stubbornness, and indecision from leadership approaches. Software development programs require involved, trustworthy, and supportive leaders setting the example.

Ilies et al. (2006) observed leadership is not about the direction of individuals. Leaders motivate, support, inspire, and foster achievement in themselves and others (Ilies et al.). Demonstrating enthusiasm for objectives, providing employee support, and participating in team activities are attributes of successful leadership (Northouse, 2010; Taylor, 2007). Leaders obtain the maximum potential from followers through motivation, encouragement, support, and participation (Tarabishy et al., 2005). Effective leadership inspires followers to higher performance levels through setting a positive example (Ilies et al.).

In the study, participant views revealed the importance of setting the example, working with the team, and removing ego from leadership approaches. Successful leadership for software development is a part of the solution through interaction and support of individuals and teams. Leaders participating and supporting team activities contribute to program success.

Emerging Theory for Leadership Environment

Analysis of data collected for questions 4, 5, 6, and 7 resulted in five themes related to leadership characteristics, attributes, and approaches for software engineering. Interpretation of the resulting themes revealed an emerging theory related to the open systems paradigm of environment, resources, and processes. The themes for software leadership relate to development of an effective environment for success. Participant views revealed software development programs result in successful outcomes through implementation of effective leadership environments.

Themes derived from the data revealed the requirement for leadership environments providing effective communication, teamwork, experience, empowerment, and leadership by example. These leadership approaches result in an effective environment for performing software development. The research data indicated the importance of implementing an effective leadership environment for software development success.

The leadership environment provides the foundation for team interaction, performance, and productivity. Integrating the critical components for an effective environment contributes to supporting team initiatives and achieving successful outcomes. Developing a successful leadership environment requires application of each of these components for success. Focusing on any single attribute limits the effectiveness of the leadership approach.

An effective leadership environment incorporates successful approaches for communication, teamwork, experience, empowerment, and examples. Integration of these components provides the environmental aspect of the emerging leadership theory. Figure 7 presents emerging theory development for the environmental component of the open system theoretical framework.

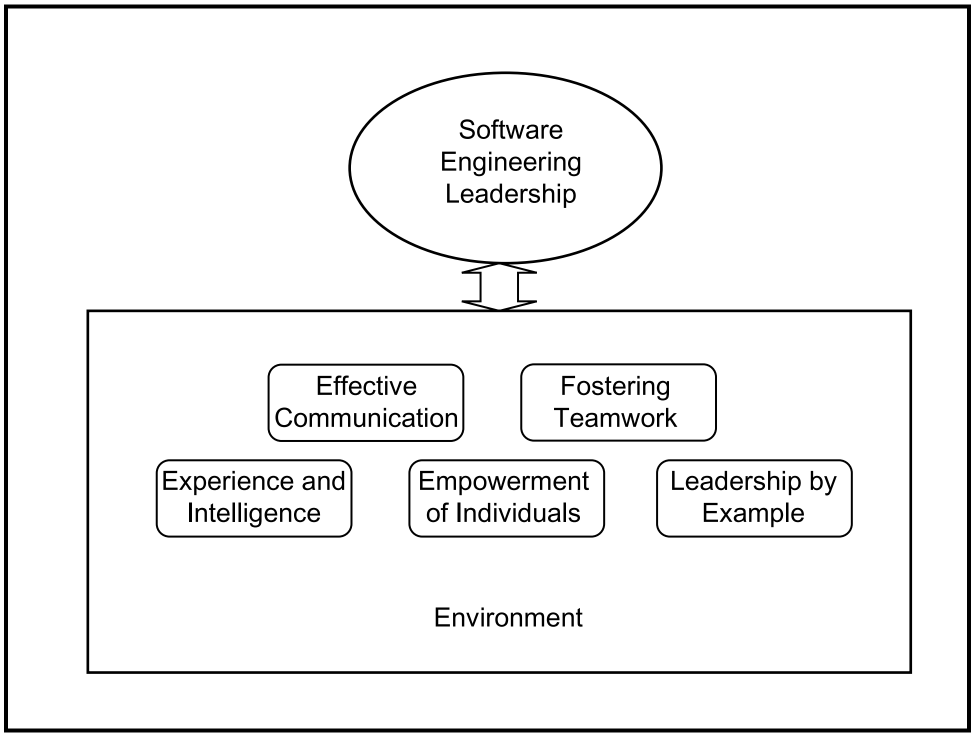

Software Engineering Leadership
Effective Communication
Fostering Teamwork
Experience and Intelligence
Empowerment of Individuals
Leadership by Example
Environment

Figure 7. Environmental component for emerging leadership theory.

The open system paradigm integrates the areas of environment, resources, and processes. Investigation of the primary research question resulted in the identification of software leadership environments contributing to successful program initiatives. The supplemental research questions resulted in identification of attributes for resources and processes.

Software Resource Category

Open-ended question 8, 9, and 10 of the electronic interview questionnaire sought to obtain views and perceptions on software practices, methodologies, and resources for successful programs. Question 8 investigated perceptions on the most important process for software development. Question 9 investigated perceptions on the most effective software development methodology. Question 10 investigated perceptions on ineffective software development methodologies.

Data coding and analysis for questions 8, 9, and 10 revealed core themes related to required resources for successful software development. Participant responses resulted in four main themes related to required resources for successful development programs: (a) requirements definition and management, (b) established procedures and processes, (c) effective planning and scheduling, and (d) adequate testing. Table 14 provides a summary of the frequency of themes in the participant responses.

Table 14

Frequency of Resource Themes for Participant Responses

Theme	Frequency
Requirements Definition and Management	48
Established Procedures and Processes	42
Effective Planning and Scheduling	26
Adequate Testing	10

These four themes relate to the approaches and resources required for software development programs. Each of these themes provides a critical resource for conducting software development programs. Participants stressed the importance of these resources in achieving successful outcomes.

Requirements definition and management. Requirements definition and management was the predominant resource identified for successful development programs. Participants referenced the importance of requirements 48 times in the responses. Participants stressed the importance of requirements management for successful software programs. Participants referenced ineffective requirements management 7 times as a contributing factor to failed software programs. Developing accurate requirements is a key resource for successful software development programs.

Requirements define the desired functionality for the software product (Pressman, 2010). Defining and managing requirements is a critical activity contributing to the success of the software product. Implementing requirements management provides a framework for design, development, and testing. Assuring customer and end user requirements are met results in increased customer satisfaction, improved product acceptance, and reduced errors (Salinas, Prudhomme, & Brissaud, 2008).

Requirements management assists the development team in controlling, identifying, tracking, and managing changes throughout the project life cycle (Pressman, 2010). Failure to manage requirements results in uncontrolled, continually changing, and hard to manage software programs (Pressman; Salinas et al., 2008). Incorporating effective requirements management approaches supports successful software development.

Established procedures and processes. The second most frequent resource listed was establishing and following procedures and processes for the program. Participants referenced the importance of established procedures 42 times. Providing procedures and processes for the development life cycle is a critical resource for software development. The generation of procedures and processes for software development activities is critical to program success.

Software processes support the effective and organized development of software products (Jianguo et al., 2008). Through application of software development and management processes, the quality of products is enhanced and development efficiency is improved (Li, Chen, & Lee, 2003). The performance of the organization depends on successful software development. Establishing and following software development and management processes throughout the life cycle supports increased productivity and improved performance (Jianguo et al.). Software processes provide the foundation for successful execution of tasks and development activities to achieve software engineering objectives.

Effective planning and scheduling. Establishing comprehensive plans and schedules for program activities was referenced 26 times in participant responses. The importance of developing and following plans and schedules to meet critical objectives was a core theme in the data. Program plans and schedules are key resources for effective software engineering programs.

Effective planning is essential to identifying approaches, challenges, and opportunities for achieving program objectives (Amiri, Kavousy, & Azimi, 2010). Developing and implementing program plans and schedules provides the foundation for successful task execution. Leaders for software development programs establish program plans, define development schedules, and monitor activities throughout the product development life cycle (Kerzner, 2009). Plans and schedules provide a method of benchmarking performance, identifying potential shortfalls, and supporting resource allocation (Amiri et al., 2010). Establishing, monitoring, and following plans and schedules provide leadership with the roadmap for successful software development.

Adequate testing. Participants cited the importance of adequate test approaches in the responses 10 times. Adequate testing of developed products is a critical resource in determining the success of development efforts. Delivery of products without adequate testing results in failed products, programs, and reduced customer satisfaction.

The success of software development programs depends on user acceptance of the delivered product. Delivering accurate and reliable software requires adequate testing (Iacob & Constantinescu, 2008). Testing verifies the system meets established requirements, the product performs as designed, and critical errors are identified before delivery (Pfleeger & Atlee, 2010). Software programs are often listed as failures when the product does not meet user requirements (Rubinstein, 2007; Woolridge et al., 2009). Performing adequate testing results in improved product quality, increased reliability, and enhanced user satisfaction (Iacob & Constantinescu). Establishing adequate testing approaches supports successful software development initiatives.

Emerging Theory for Software Development Resources

Analysis of data collected for questions 8, 9, and 10 resulted in four themes related to leadership resources for successful software development programs. Interpretation of the resulting themes revealed an emerging theory related to the open systems paradigm of environment, resources, and processes. The themes for effective software practices relate to resources contributing to successful software development. Participant views revealed software development programs result in successful outcomes through inclusion of resources for software development.

Themes derived from the data revealed the requirement for software development resources resulting in requirement definition and management, established procedures and processes, effective planning and scheduling, and adequate testing. Providing these software engineering specific resources results in successful software development. The research data emphasized the importance of defining effective software development resources for success.

The software development resources provide the framework for performing activities throughout the development life cycle. Developing a set of critical resources for successful software development fosters improved performance and increased product quality. Developing a successful leadership approach to software development requires awareness of these required software resources. Effective application of resources at each stage of development results in achievement of program objectives, quality products, and user satisfaction.

Leadership for success software programs incorporate effective approaches for application of requirements management, processes, planning, and testing. These approaches for software activities provide the resource aspect of the emerging leadership theory. Figure 8 presents emerging theory development for the resource component of the open system theoretical framework.

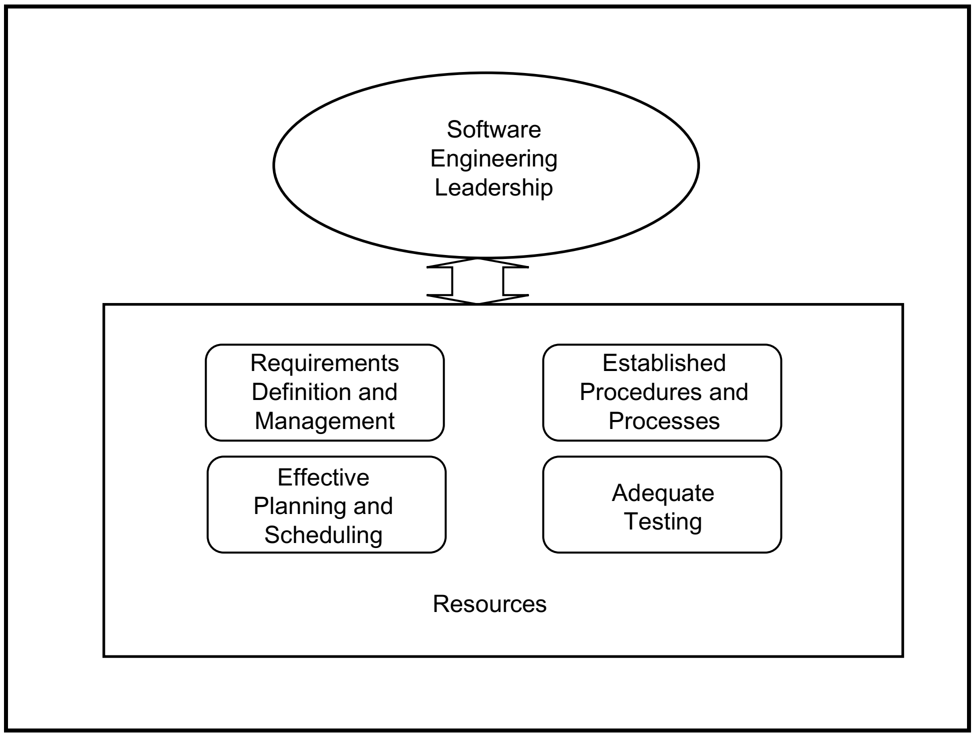

Software Engineering Leadership
Requirements Definition and Management
Established Procedures and Processes
Effective Planning and Scheduling
Adequate Testing
Resources

Figure 8. Resource component for emerging leadership theory.

Investigation of the secondary research questions resulted in the identification of software engineering resources contributing to successful program development. Software resources provide the second component for effective leadership for software development programs.

Software Process Category

Question 9 investigated perceptions on the most effective software development methodology. Question 10 investigated perceptions on ineffective software development methodologies. Participants listed agile or iterative methodologies as most effective with 21 and 18 occurrences. CMM/CMMI and waterfall methodologies were also listed at 6 occurrences each. Agile and SCRM methodologies were the most popular with rationale most often reported as previous experience. Iterative methodologies include incremental, spiral, and phased approaches to software development. Participants cited previous experience with these methodologies and successful programs as the rationale for selecting the individual method.

Data coding and analysis for questions 8, 9, and 10 revealed core themes related to required processes for successful software development. Participant responses resulted in four main themes related to processes for successful development programs: (a) agile development methodologies, (b) iterative or incremental development, (c) CMM/CMMI processes, and (d) waterfall-based methodologies. Table 15 provides a summary of the frequency of themes in the participant responses.

Table 15

Frequency of Process Themes for Participant Responses

Theme	Frequency
Agile Development Methodologies	21
Iterative or Incremental Development	18
CMM/CMMI Processes	6
Waterfall-Based Methodologies	6

These four themes relate to the processes contributing to successful software development programs. Each of these themes provides a successful process identified for software development programs. Participants stressed the importance of these components in achieving successful outcomes.

Agile development methodologies. The interview questionnaire investigated participant views on effective software development methodologies. Participants referenced the success of agile development methodologies 21 times in the responses.

Participants noted agile development resulted in successful program implementations for large and small development efforts.

Agile development methods improve software development performance through the implementation of methods for quick response to changing requirements and environments (Aken, 2008). Short development iterations produce partial functionality for testing and evaluation (Pozgaj et al., 2007). Feedback from the end user is incorporated into the following iteration to improve the product (Clutterbuck et al., 2009). Agile development methodologies provide software engineers flexibility in the development approach and result in improved customer satisfaction (Keston, 2008).

Iterative or incremental development. The second most frequent theme for software processes was iterative or incremental development. Participants referenced iterative, spiral, or incremental approaches 18 times. Iterative approaches provide developers the capability to develop partial functionality and improve the product incrementally.

Iterative, incremental, or spiral development provide overlapping increments for implementation of software development in cycles as opposed to completing an entire phase before proceeding into the next development phase (Guntamukkala et al., 2006; Rajlich, 2006). Iterative or incremental development divides the project into a series of activities represented by a traditional waterfall model (Siddiqui et al., 2006). Software development teams perform successive refinements during the life cycle to produce the product (Siddiqui et al.). The customer evaluates the product incrementally to provide feedback for product improvements (Pressman, 2010). Spiral models follow the phased approach of incremental models with additional emphasis on risk analysis and mitigation at each stage of product development (Boehm, 1988; Hashmi & Baik, 2007).

CMM/CMMI processes. Participants referenced CMM/CMMI processes as the most successful 6 times in the responses. Establishing processes through the philosophy of CMM/CMMI provides a framework for software development. Participants revealed CMM/CMMI approaches provide a structured process approach to development that supports implementation of multiple methodologies and processes for software development.

CMM/CMMI provides an approach for analyzing and understanding the capability maturity of applied processes within an organization (CMU/SEI, 2006). The CMM and CMMI approaches provide a framework for organizations to determine maturity level, identify critical issues for process improvement, define the software process, and implement process improvement programs (Galin & Avrahami, 2006). CMM/CMMI models do not define a particular process for software development but rather emphasize the importance of process implementation and continual process improvement (Jianguo et al., 2008; McManus & Wood-Harper, 2007b).

Waterfall-based methodologies. Participants referenced waterfall-based methodologies as the most successful 6 times in the responses. Participants noted waterfall approaches are successful when properly implemented. Successful waterfall implementations require effective communication and team interaction to assure success for development programs.

The waterfall model provides a sequence of phases for software development encompassing design, development, and requirement analysis (Harris et al., 2007;

Larman & Basili, 2003). Each stage implements a complete phase of the development life cycle such as design, code, test, and implementation (Harris et al.). Once a stage is completed, the team moves to the next stage of development. Each stage is completed before proceeding into the next development phase. In this approach, all phases are completed before the product is available for testing and evaluation (Harris et al.). In this method the approach to software development is highly structured and centered on completing individual milestones prior to full project completion (Harris et al., 2007; Sommerville, 2007).

Emerging Theory for Software Development Processes

Analysis of data collected for questions 8, 9, and 10 resulted in four themes related to leadership processes for successful software development programs. Interpretation of the resulting themes revealed an emerging theory related to the open systems theory of environment, resources, and processes. Participant views revealed software development programs result in successful outcomes through application of defined software processes and procedures.

Themes derived from the data revealed the four prominent software development processes of agile, iterative, CMM/CMMI, and waterfall support successful program outcomes. Following one of these processes or methodologies results in successful software development. The research data revealed these four approaches are considered the most effective when applied to software development programs.

The software development processes provide the guidelines for product design, development, and implementation. Applying approaches identified as effective in previous programs can increase program success. Developing a collection of effective approaches for selection supports development of diverse products in multiple environments. The collection of effective processes provides leadership with a set of approaches for application depending on the type of product and goals for the software program. Figure 9 presents emerging theory development for the process component of the open system theoretical framework.

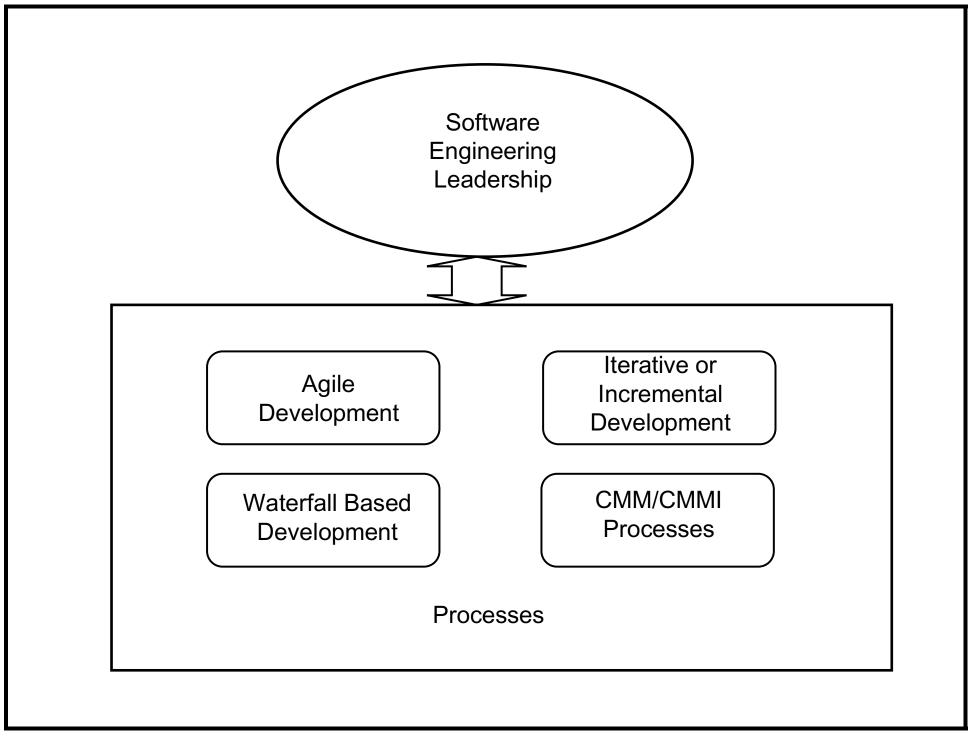

Software Engineering Leadership
Agile Development
Iterative or Incremental Development
Waterfall Based Development
CMM/CMMI Processes
Processes

Figure 9. Process component for emerging leadership theory.

Investigation of the secondary research questions resulted in the identification of software processes contributing to successful program development. Leadership environment, software resources, and software processes provide the components necessary for successful software development.

Theory for Effective Leadership for Software Development Programs

This qualitative research study investigated the leadership practices, characteristics, and approaches contributing to successful software development programs. Analysis of the participant responses resulted in the identification of five themes related to leadership environment, four themes related to resources, and four themes related to software processes. Emerging theory for each of the components of the open system paradigm revealed core themes for environment, resources, and processes. The individual theories for each of these components can be combined into a theory on the environment, resources, and process required for successful software development programs.

Question 10 of the study investigated participant views on development methodologies that are ineffective. For this question, respondents did not identify specific ineffective processes; instead participants stated the issue is related to management of programs and not specific processes applied. Participants stated the software methodology is not the reason for program failure, rather ineffective management results in program failure. Several participants noted any process can be successful with effective leadership. Additionally, failures of programs are not the result of ineffective processes, but rather the result of ineffective management.

Respondent views on the importance of management approach provide the foundation for the software development leadership theory. The responses indicate processes and resources provide the framework for successful program approaches and effective leadership is the key to success. Emerging theory for the environment, resource, and process components can be integrated to develop a theory on effective software leadership.

Effective software leadership begins with establishment of an environment encouraging individual, team, and organizational success. The leadership theory identifies the critical components of communication, teamwork, empowerment, intelligence, and leading by example are required to establish an effective environment for software development. These components provide the foundation for successful leadership. Without an effective environment, the application of process and resources will not result in successful software development. The foundation of the theory is the establishment of an effective environment.

Establishing an effective environment for software development is only part of the theory of successful leadership. Along with the environment, the framework for success must be established. The framework identifies resources and processes essential to successful development. The software resources provide a collection of essential techniques and components required for program success. Requirements management, established processes, effective planning, and adequate testing are essential to achieving program success. Leadership for software development must maintain awareness of effective software resources for development. Application of selected resources supports achievement of objectives and development of successful software products.

Establishing a successful environment and defining resources for success provide two dimensions of the leadership theory. The third area is the identification of effective processes. The discipline of software engineering contains numerous approaches and processes. Application of the appropriate processes supports successful software development. Identification of approaches resulting in success on previous programs provides a collection of methodologies for leadership application. The identified approaches resulting in successful initiatives include iterative, agile, waterfall, and

CMM/CMMI methodologies. The collection of processes provides leadership with a set of methods for application.

The three components are environment, resource, and processes are combined into a model for successful software development leadership. The three components work together to provide leadership with an integrated set of concepts and methodologies supporting development of reliable and quality software. The environment component establishes a foundation for increased productivity and performance. The resource and process components provide a set of techniques supporting successful software development. Leadership achieves success in software through integration and application of the identified techniques and processes. Figure 10 provides the theory developed from the data and theoretical framework of open systems.

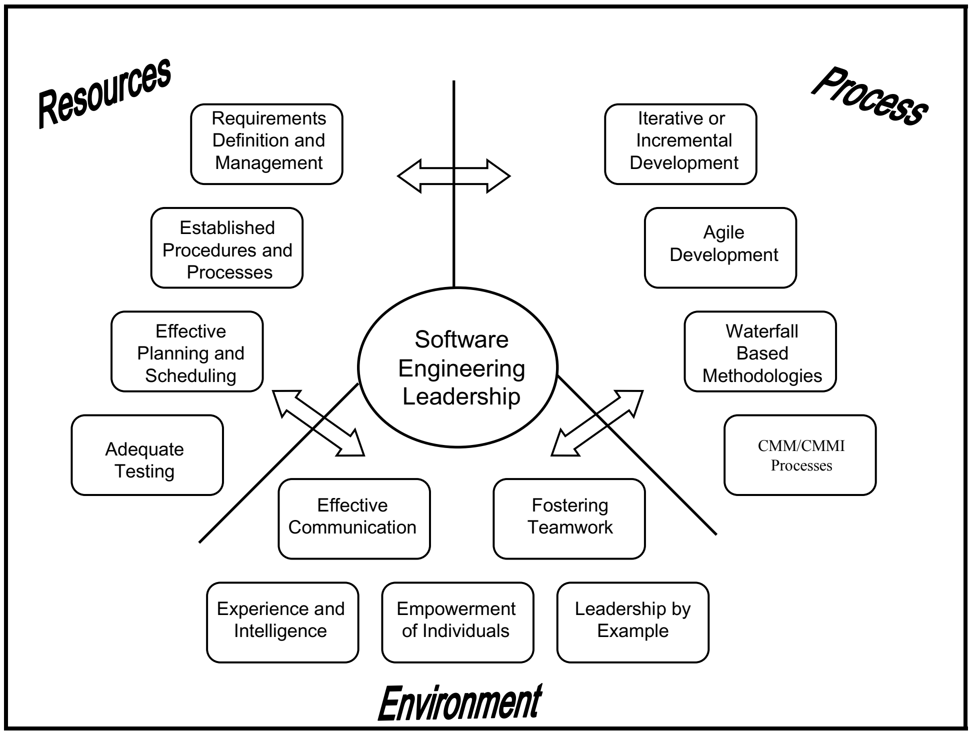

Resources
Process
Requirements Definition and Management
Established Procedures and Processes
Effective Planning and Scheduling
Adequate Testing
Iterative or Incremental Development
Agile Development
Waterfall Based Methodologies
CMM/CMMI Processes
Software Engineering Leadership
Effective Communication
Fostering Teamwork
Experience and Intelligence
Empowerment of Individuals
Leadership by Example
Environment

Figure 10. Theory for successful software development leadership.

The software engineering leadership theory emphasizes the importance of the leadership environment for success. Environment provides the foundation for successful project approaches. The components of resource and process provide the framework for performing software development in the established leadership environment. With successful management and leadership approaches, the applied resources and processes contribute to program success.

Uniqueness of Findings

This qualitative grounded theory study resulted in the development of a theory for successful software engineering leadership. Analysis of the data revealed key concepts for leadership environment, software resources, and software processes. The leadership environment identifies five critical areas for success. These areas are not unique to the software engineering field but reflect critical components for leadership success identified in previous leadership theory and program management methods. Participant views and perceptions did not identify unique leadership approaches required for software engineering. The critical components for success identified for leadership, program management, and change management initiatives are similar to the leadership environment components identified in this study.

The unique components for software leadership are in the integration of resources and processes. Identifying process for software success alone is not sufficient, the processes and critical software resource areas must be integrated into the program leadership philosophy. Selecting the appropriate combination of processes and assuring the leadership environment provides the foundation for success throughout the software life cycle provides the combination of attributes for success.

Limitations

The study scope was limited to research and development organizations in Alabama and the study results reflect individual perceptions and opinions based on the operations in the localized area, economic impacts, and job stability. The study did not generalize the results to varying populations or geographic locations. The findings of the study are not generalized to the population but rather are applicable to a particular phenomenon (Bogdan & Biklen, 2007).

The study was also limited by the use of an Internet interview questionnaire to collect participant responses. The electronic interview questionnaire provided data from participants but did not support additional probing questions. The responses provided were the only data available for analysis.

Implications

The development of complex software applications requires leadership supporting the software development life cycle (Sommerville, 2007). Successful software development requires implementation of an innovative and creative environment for success. Software resides in many products used by consumers (Stackpole, 2008) and advanced applications (Schneidewind, 2007). The development of software applications represents significant investments for organizations. Assuring successful program development through effective leadership results in improved proficiency, enhanced performance, and increased profits.

Leadership for software engineering programs strives to provide products that meet consumer and organizational requirements. Identification of effective leadership approaches and software processes can result in improved software development programs (Tesch et al., 2007). Software development organizations require effective leadership for development programs to support consumer demands, industry standards, and organizational objectives (Hadar & Leron, 2008; King, 2007; Schneidewind, 2007).

The success of software development programs can be improved through the application of effective leadership approaches. Implementing leadership theories and models results in improved program success. The leadership theory for successful software development provides a model for establishing an effective environment and defining required software resources and processes. This approach provides a technique for successful software development leadership.

This qualitative grounded theory study investigated leadership and software development practices for successful programs. Participant views and perspectives on software leadership resulted in the identification of leadership theory for software development. Effective leadership environments provide communication, teamwork, experience, empowerment, and leadership by example. Software resources and processes for success encompass requirements management, implemented processes, effective planning, and adequate testing. Integrating effective leadership attributes and software development approaches results in successful software development.

The data analysis resulted in the identification of common themes for software development leadership. The identified themes are also key drivers for leadership and change management initiatives and are not specific to the field of software engineering. Successful leadership for software engineering does not require unique approaches to leadership. The results indicate sound leadership practices contribute to successful projects and do not require unique approaches for the field of software engineering. To improve the performance and success of software development initiatives, executive management should focus on developing effective leaders following established leadership concepts. Software engineering does not require unique approaches for leadership but rather, effective leadership supports successful software engineering.

The research results indicate improvement in software development performance requires the application of a combination of effective resources, procedures, and processes. Applying an individual approach may improve a specific area of software development but continued success for the software program requires application of several processes and procedures throughout the software life cycle. Success in software development requires a combination of successful leadership, procedures, process, and approaches. The research did not identify a best approach instead the research indicated a combination of approaches throughout the lifecycle resulted in improved program performance.

The research study investigated views from leadership and software development experts related to software development programs and processes. The results present unexpected results for leadership and software processes. Surprising results from the study are: (a) failure is a result of leadership not processes, (b) no discussion of model driven development, (c) waterfall methodology in use at research organization, (d) CMM/CMMI not a major theme.

A predominant theme in the data is the importance of leadership for successful software development. Question 10 investigated participant view on ineffective software development processes. The major response for this question centered on effective leadership. Participants stated leadership is the key to successful software development not processes.

In question 11, participants were given the opportunity to provide any additional comments on leadership and software engineering. Respondents emphasized the importance of effective leadership over process. The approaches and methodologies applied are not the cause of software failure. Participant views revealed software failures are the result of inadequate or ineffective leadership.

Model driven development is a predominant methodology for software development. Models define the system product at each stage of development resulting in graphical representations for implementation (Pressman, 2010). This approach is used throughout the selected research organization for software development. However, only one response included discussions of model driven development. In discussions on ineffective methodologies, participant 28 noted experience with model driven development resulting in software failure. Model driven development approaches or derivatives were not included in other participant responses.

The waterfall methodology emerged in the 1970s as an approach to meet stringent government contracting requirements for software development (Harris et al., 2007; Larman & Basili, 2003). The rigid and disciplined approached to software engineering through the waterfall methodology resulted in experts favoring more fluid and innovative methods (Harris et al.; Sommerville, 2007).

Responses to the interview questionnaire indicate the waterfall method is still in use at the selected research organizations. Participants provided positive responses on the waterfall methodology seven times and negative responses eight times. The responses indicate mixed results on the success of waterfall methodologies. Further research is needed to identify the approaches that result in successful programs when the waterfall methodology is applied. The success of waterfall methodologies may depend on the selected implementation, planning, and leadership.

The selected organization for the research study has attained a CMMI Level IV rating. Participant responses were expected to include information on CMM/CMMI approaches and methodologies as a major theme due to the familiarity and predominance in the organization. However, for the 71 participants only seven provided positive comments and two provided negative comments on CMM/CMMI. This response indicates the CMM/CMMI methodology is not a common approach in the organization. The application of this methodology may be successful if combined with an effective leadership environment and software resource concepts.

Recommendations

This study investigated leadership characteristics and software development processes that contribute to successful programs.

Recommendations for Leadership

The data analysis resulted in the identification of successful approaches for leadership, software resources, and software processes. Recommendations to improve software development efforts are:

1. Integration of leadership training programs for key software engineering experts. Understanding the criticality of leadership processes supports an integrated approach to software life cycle efforts.

2. Development of a software resource and software process tool kit to achieve a collective set of successful processes. Define specific activities that contribute to success and provide recommendations for types of programs and phases for application.
3. Develop lessons learned and share results throughout the organization. Significant information is obtained from past performance of successful and failed programs. Provide a repository for lessons learned to support improvement in current initiatives and eliminate know problematic approaches.
4. Incorporate leadership training in software engineering college curricula. Understanding the phases, processes, and approaches for software engineering is not sufficient to assure success. Software engineers must also understand the importance of effective leadership approaches for software efforts. Providing leadership training early in the development of future software engineers may improve the success rate of software development efforts.

Recommendations for Future Research

The study developed qualitative grounded theory on successful leadership environments, resources, and processes for success. Qualitative research generates theory from emerging themes and may be enhanced through a quantitative study on a similar topic (Glaser & Strauss, 2007). The study results may be further enhanced through a quantitative study investigating similar concepts.

The research study investigated views and perceptions from leadership and software developers at one organization. Future research may enhance the results through investigation of additional perspectives. Suggestions for future research include:

1. A research study that investigates multiple organizations to obtain different perspectives outside of the organizational paradigm.
2. A research study at an organization without a CMM/CMMI rating. The identification of successful leadership approaches and software development processes may result in additional information on effective environments, resources, and processes.
3. A research study investigating the leadership and software processes applied during development of different criticality levels. Software processes and leadership approaches may be different when developing software considered safety critical versus non-safety critical.

Summary

The purpose of this qualitative grounded theory study was to investigate leadership and development practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes. The study explored the experiences and unique perceptions of leaders and software developers involved in software development programs. The selected research organization provided participants with experience and backgrounds in leadership and software engineering. An electronic interview questionnaire obtained participant responses on effective leadership approaches and software development processes.

The interview questionnaire investigated the primary research question on leadership practices and obtained views and perceptions on successful leadership characteristics and approaches for software development programs. Data coding and analysis resulted in five common themes for successful leadership: (a) effective communications, (b) fostering teamwork, (c) experience and intelligence, (d) empowerment of individuals, and (e) leadership by example. Data analysis resulted in four themes for leadership resources: (a) requirements definition and management, (b) established procedures and processes, (c) effective planning and scheduling, and (d) adequate testing. Four themes emerged on the required processes for successful software development: (a) agile development methodologies, (b) iterative or incremental development, (c) CMM/CMMI processes, and (d) waterfall-based methodologies.

Review of the themes provided in the data resulted in the development of a leadership theory for successful software development in the open system framework. Software processes and resources provide the framework for successful program approaches and effective leadership is the key to success. The themes on environment, resource, and process components are integrated to develop a theory on effective software leadership.

The foundation for effective leadership is the established environment for success. The leadership theory integrates the critical components for implementing a successful software development environment. The leadership theory stresses the importance of effective communication, fostering teamwork, empowerment, intelligence, and leadership by example. Integrating these components provides a foundation for development of an effective leadership environment for successful software development.

The leadership theory implements the software resource component to support a framework for effective development. The critical resources for software development implement approaches for requirements management, process application, program planning, and system testing. Application of these software resources results in improved performance and enhanced product quality. The leadership theory integrates the resource component to provide leadership with a set of approaches for performing successful software development.

The third component of the leadership theory identifies successful software development processes. The processes of incremental development, agile methodologies, CMM/CMMI philosophy, and waterfall development provide a set of approaches for life cycle development. The process component of the leadership theory provides leadership identified methodologies for success. Leadership application of the identified approaches provides a framework for successful development of software products.

The leadership theory for successful software development integrates techniques for establishing an effective environment, implementing software resources, and applying software processes to support software programs. Application of the developed theory and identified techniques can result in improved product development and increased product quality. Leadership environment provides the critical capabilities, attributes, and concepts for successful project development. The resources and processes support leadership in achieving development objectives.

Developing new approaches and applying new theories can improve the existing software development success rate. An effective leadership approach to software development results in increased performance, improved productivity, and enhanced user satisfaction. The leadership theory fosters improved performance through the integration of environment, resources, and processes.

Organizations can apply the leadership theory for successful software development to establish effective environments, implement successful processes, and incorporate efficient resources. The leadership theory identifies a set of components and concepts that when integrated and applied can result in improved software development performance. Implementation of the theory for successful software development leadership can be used to improve organizational performance through enhanced leadership and software development approaches.

REFERENCES

Adams, T. L. (2008). Interprofessional relations and the emergence of a new profession: Software engineering in the United States, United Kingdom, and Canada. *Sociological Quarterly*, *48*(3), 507-532. doi: 10.111/j.1533-8525.2007.00087.x

Agerfalk, P. J., & Fitzgerald, B. (2006). Flexible and distributed software processes: Old petunias in new bowls. *Communications of the ACM*, *49*(10), 27-34. Retrieved September 3, 2009, from EBSCOhost database.

Agrawal, M., & Chari, K. (2007). Software effort, quality, and cycle time. A study of CMM level 5 projects. *IEEE Transactions on Software Engineering*, *33*(3), 145-156. Retrieved August 8, 2009, from ProQuest database.

Aken, A. (2008). CHUNK: An agile approach to the software development life cycle. *Journal of Internet Commerce, 7*(3), 313-338. Retrieved January 24, 2010, from EBSCOhost database.

Alam, M., Hafner, M., & Breu, R. (2008). Constraint based role based control in the SECTET-framework. *Journal of Computer Security*, *16*(2), 223-260. Retrieved August 28, 2009, from EBSCOhost database.

Al-Qutaish, R. E., & Al-Sarayreh, K. (2008). Software process and product ISO standards: A comprehensive survey. *European Journal of Scientific Research*, *19*(2), 289-303. Retrieved August 25, 2009, from EBSCOhost database.

Amiri, S. R. S., Kavousy, E., & Azimi, S. Y. (2010). The role of cultural strategic planning in increasing organizational productivity, development and perfection. *European Journal of Social Sciences, 15*(2), 69-74.

Aoumeur, N. (2008). Stepwise rigorous development of distributed agile information systems: From UML-diagrams to component-based Petri nets. *Enterprise Information Systems*, *2*(2), 125-160. doi: 10.1080/17515700801927403

Awazu, Y., Baloh, P., Desouza, K. C., Wecht, C. H., Kim, j., & Jha, S. (2009). Information – Communication technologies open up innovation. *Research Technology Management, 52*(1), 51-58. Retrieved March 7, 2009, from EBSCOhost database.

Babbie, E. (2010). *The practice of social research* (12th ed.). Belmont, CA: Wadsworth.

Balasubramanian, K., Gokhale, A., Lin, Y., Ahang, J., & Gray, J. (2006). Weaving deployment aspects into domain-specific models. *International Journal of Software Engineering and Knowledge Engineering*, *16*(3), 402-424. Retrieved August 28, 2009, from EBSCOhost database.

Baillie, L., Ford, P., Gallagher, A., & Wainwright, P. (2009). Nurses' views on dignity in care. *Nursing Older People, 21*(8), 22-29. Retrieved January 21, 2010, from EBSCOhost database.

Basili, V. R., Cruzes, D., Carver, J. C., Hochstein, L. M., Hollingsworth, J. K., Zelkowitz, M. V., et al. (2008). Understanding the high-performance computing community: A software engineer's perspective. *IEEE Software, 25*(4), 29-36. Retrieved August, 28, 2009, from ProQuest database.

Basili, V. R., & Zelkowitz, M. V. (2007). Empirical studies to build a science of computer science. *Communications of the ACM, 50*(11), 507-532. Retrieved October 21, 2009, from EBSCOhost database.

Bass, B. M. (1999). Two decades of research and development in transformational leadership. *European Journal of Work and Organizational Psychology*, *8*(1), 9-32. doi: 10.1080/135943299398410

Beadell, B. (2009). CMMI as contemporary iron case: A grounded analysis from the perspective of practicing engineers in defense engineering. (Doctoral dissertation, University of St. Thomas Minnesota, 2009). Retrieved March 6, 2010, from ProQuest Dissertations and Theses: Full Text. (Publication No. AAT3388217).

Bell, A. E. (2008). From the front lines software development amidst the whiz of silver bullets. *Communications of the ACM, 51*(8), 22-24. Retrieved October 20, 2009, from EBSCOhost database.

Benediktsson, O., Dalcher, D., & Thorbergsson, H. (2006). Comparison of software development life cycles: A multiproject experiment. *IEE Proceedings - Software*, *153*(3), 87-101. doi: 10.1049/ip-sen:20050061

Berg, B. L. (2009). *Qualitative research methods for the social sciences* (7th ed.). Boston, MA: Allyn and Bacon.

Bernstein, P. A., & Haas, L. M. (2008). Information integration in the enterprise. *Communications of the ACM*, *51*(9), 72-79. doi: 10.1145/1378727.1378745

Bharadwaj, S. S., & Saxena, K. B. (2006). Impacting the process of global software teams: A communication technology perspective. *VISION - The Journal of Business Perspective*, *10*(4), 63-75. Retrieved August 20, 2008, from EBSCOhost database.

Bird, C., Nagappan, N., Devanbu, P., Gall, H., & Murphy, B. (2009). Does distributed development affect software quality? An empirical case study of windows vista. *Communications of the ACM*, *52*(8), 85-93. doi: 10.1145/1536616.1536639

Bixenman, M. L. (2007). Leading open innovation across global strategic alliances: A grounded theory study. (Doctoral dissertation, University of Phoenix, 2008). Retrieved March 9, 2010, from ProQuest Dissertations and Theses @ University of Phoenix. (Publication No. AAT 3289586).

Bloch, D. P. (2008). Complexity, connections, and soul work. *Catholic Education: A Journal of Inquiry and Practice*, *11*(4), 543-554. Retrieved June 18, 2009, from EBSCOhost database.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, *21*(5), 61-72. Retrieved August 26, 2009, from IEEE Computer Society Digital Library database.

Boehm, B. W. (2006). A view of 20th and 21st century software engineering. *Proceedings of the 28th International Conference on Software Engineering*, 12-29. Retrieved August 20, 2009, from ACM Digital Library database.

Boehm, B. W., & Valerdi, R. (2008). Achievements and challenges in Cocomo-based software resource estimation. *IEEE Software*, *25*(5), 74-83. Retrieved July 31, 2009, from ProQuest database.

Boerner, S., Eisenbeiss, S. A., & Griesser, D. (2007). Follower behavior and organizational performance: the impact of transformational leaders. *Journal of Leadership and Organizational Studies*, *13*(3), 15-26. Retrieved September 6, 2009, from EBSCOhost database.

Bogdan, R. C., & Biklen, S. K. (2007). *Qualitative research for education: An introduction to theory and methods* (5th ed.). New York: Pearson Education, Inc.

Bonner, N. A. (2008). Acceptance of systems development methodologies: Testing a theoretically integrated model. (Ph. D. dissertation, The University of Texas at Arlington, 2008). Retrieved March 6, 2010, from ProQuest Dissertations and Theses: Full Text. (Publication No. AAT3320052).

Booch, G. (2008). Morality and the software architect. *IEEE Software*, *25*(1), 8-9. Retrieved August 6, 2009, from ProQuest database.

Bose, I. (2008). Lessons learned from distributed agile software projects: A case-based analysis. *Communication of the Association for Information Systems*, *23*, 619-632. Retrieved September 3, 2009, from EBSCOhost database.

Boseman, G. (2008). Effective leadership in a changing world. *Journal of Financial Service Professionals*, *62*(3), 36-38. Retrieved September 8, 2009, from EBSCOhost database.

Boynton, B. C. (2007). Identification of process improvement methodologies with application in information security. *Proceedings of the 4th Annual Conference on Information Security Curriculum Development*, *28*. Retrieved August 31, 2009, from ACM Digital Library database.

Brown, A. W., & McDermid, J. A. (2008). The art and science of software architecture. *International Journal of Cooperative Information Systems*, *16*(3/4), 439-466. Retrieved August 9, 2009, from EBSCOhost database.

Cagle, W. (2007). Real change leadership in twenty-first century missions context. *Asian Journal of Pentecostal Studies*, *10*(1), 63-77. Retrieved September 8, 2009, from EBSCOhost database.

Calabrese, D. (2008). *Project management best practices* (00011206). Faulkner Information Services. Retrieved from Faulkner's Advisory on Computers and Communications Technologies

Cantor, M. (2002). *Software leadership: A guide to successful software development.* Indianapolis, IN: Addison-Wesley.

Carnegie Mellon University Software Engineering Institute. (2006). *CMMI for development, version 1.2: Improving processes for better products*. (CMU/SEI-2006-TR-008 ESC-TR-2006-008). Pittsburgh, Pennsylvania: Author.

Cerpa, N., & Verner, J. M. (2009). Why did your project fail? *Communications of the ACM, 52*(12), 130-134. doi:10.1145/1610252.1610286

Chan, Y. H., Taylor, R. R., & Markham, S. (2008). The role of subordinates' trust in a social exchange-driven psychological empowerment process. *Journal of Managerial Issues*, *20*(4), 444-467. Retrieved September 8, 2009, from EBSCOhost database.

Charmaz, K. (2006). *Constructing grounded theory: A practical guide through qualitative analysis.* Thousand Oaks, CA: Sage Publications Inc.

Chatterjee, S. (2008). Software engineering practice in computer science courses. *19th Australian Conference on Software Engineering*, 611-616. doi: 10.1109/ASWEC.2008.58

Claiborne, C. B. (2007). Innovation: A necessity of the new global business paradigm. *International Journal of Business Research*, *7*(6), 73-76. Retrieved June 17, 2009, from EBSCOhost database.

Clutterbuck, P., Rowlands, T., & Seamons, O. (2009). A case study of SME web application development: Effectiveness via agile methods. *Electronic Journal of Information Systems, 12*(1), 13-26. Retrieved January 24, 2010, from EBSCOhost database.

Colbert, A. E., Kristof-Brown, A. L., Bradley, B. H., & Barrick, M. R. (2008). CEO transformational leadership: The role of goal importance congruence in top management teams. *Academy of Management Journal, 51*(1), 81-96. Retrieved January 24, 2010, from EBSCOhost database.

Corbin, J., & Strauss, A. (2008). *Basics of qualitative research* (3rd ed.). Thousand Oaks, CA: Sage Publications.

Creswell, J. W. (2008). *Educational research: Planning, conducting, and evaluating quantitative and qualitative research* (3rd ed.). Upper Saddle River, NJ: Pearson.

Cusumano, M. A. (2008). Managing software development in globally distributed teams. *Communications of the ACM*, *51*(2), 15-17. Retrieved October 20, 2008, from EBSCOhost database.

Dalcher, D., & Benediktsson, O. (2006). Managing software development project size: Overcoming the effort-boxing constraint. *Project Management Journal*, *37*(2), 51-58. Retrieved July 30, 2009, from EBSCOhost database.

Denning, S. (2007). *The secret language of leadership*. San Francisco: Jossey-Bass.

Desouza, K. C., Awazu, Y., & Baloh, P. (2006). Managing knowledge in global software development efforts: Issues and practices. *IEEE Software*, *23*(5), 30-37. Retrieved August 25, 2009, from ProQuest database.

Dey, P. K., Kinch, J., & Ogunlana, S. O. (2007). Managing risk in software development. *Industrial Management and Data Systems*, *107*(2), 284-303. doi: 10.1108/02635570710723859

Douglas, I. (2006). Issues in software engineering of relevance to instructional design. *TechTrends*, *50*(5), 28-35. Retrieved November 13, 2008, from ProQuest database.

Early, M. M. (2006). Improving the success rate of software development projects. (Ph.D. dissertation, Northcentral University, 2006). Retrieved March 6, 2010, from ProQuest Dissertations and Theses: Full Text. (Publication No. AAT3200346).

Ebert, C. (2008). A brief history of software technology. *IEEE Software*, *25*(6), 22-25. Retrieved August 20, 2009, from IEEE Computer Society Digital Library database.

Edwards, C. (2003). Model development. *IEE Review*, *49*(8), 42-45. Retrieved August 28, 2009, from EBSCOhost database.

El Emam, K. E., & Koru, A. G. (2008). A replicated survey of IT software project failures. *IEEE Software*, *25*(5), 84-90. Retrieved November 17, 2008, from ProQuest database.

Eldai, O. I., Hussan, A., Ali, M. H., & Raviraja, S. (2008). Towards a new methodology for developing web-based systems. *Proceedings of World Academy of Science, Engineering and Technology, 36*, 190-195. Retrieved January 23, 2010, from EBSCOhost database.

Elfatatry, A. (2007). Dealing with change: Components versus services. *Communications of the ACM, 50*(8), 35-39. Retrieved August 25, 2009, from EBSCOhost database.

Erdogmus, H. (2008). Essentials of software process. *IEEE Software*, *25*(4), 4-7. Retrieved August 25, 2009, from ProQuest database.

Erdogmus, H., & Williams, L. (2003). The economics of software development by pair programmers. *Engineering Economist*, *48*(4), 283-319. Retrieved June 6, 2009, from EBSCOhost database.

Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007). Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, *24*(1), 135-169. doi: 10.2753/MIS0742-1222240104

Fichten, C. S., Ferraro, V., Asuncion, J. V., Chwojka, C., Barile, M., Nguyen, M. N., et al. (2009). Disabilities and e-learning problems and solutions: An exploratory study. *Educational Technology and Society, 12*(4), 241-256. Retrieved January 21, 2010, from EBSCOhost database.

Fruchter, R., Swaminathan, S., Boraiah, M., & Upadhyay, C. (2007). Reflection in interaction. *AI and Society*, *22*, 211-223. doi: 10.1007/s00146-007-0121-6

Galin, D., & Avrahami, M. (2006). Are CMM program investments beneficial? Analyzing past studies. *IEEE Software*, *23*(6), 81-87. Retrieved August 25, 2009, from ProQuest database.

Gefen, D., Zviran, M., & Elman, N. (2006). What can be learned from CMMI failures? *Communications of AIS*, *17*, 2-28. Retrieved April 4, 2008, from EBSCOhost database.

Glaser, B., & Strauss, A. (1967). *The discovery of grounded theory: Strategies for qualitative research.* Chicago: Aldine.

Glaser, B., & Strauss, A. (2007). *The discovery of grounded theory: Strategies for qualitative research.* Piscataway, NJ: Aldine Transaction.

Glass, R. L. (2008). One man's quest for the state of software engineering's practice analyzing researchers' finding of software practitioners' activities. *Communications of the ACM*, *50*(5), 21-23. Retrieved August 20, 2009, from EBSCOhost database.

Gorry, B. (2008). Concerns regarding the adoption of the model driven architecture in the development of safety critical avionics applications. *Proceedings of World Academy of Science, Engineering and Technology*, *28*, 87-95. Retrieved April 29, 2009, from EBSCOhost database.

Gottesdiener, E. (2008). Good practices for developing user requirements. *Journal of the Quality Assurance Institute*, *22*(3), 15-18. Retrieved from *EBSCOhost* database.

Goztas, A., Baytekin, E. P., & Kamanlioglu, E. B. (2009). Six sigma approach in business enterprises: evidence from Schneider Electric Turkey - An assessment in terms of internal organizational communication and corporate culture. *International Journal of Management Perspectives*, *1*(3), 45-71. Retrieved August 31, 2009, from EBSCOhost database.

Greenwood, P. B., & Kanters, M. A. (2009). Talented male athletes: Exemplary character or questionable characters? *Journal of Sport Behavior, 32*(3), 298-324. Retrieved January 21, 2010, from EBSCOhost database.

Guntamukkala, V., Wen, H. J., & Tarn, J. M. (2006). An empirical study of selecting software development life cycle models. *Human Systems Management, 25*(4), 265-278. Retrieved January 24, 2010, from EBSCOhost database.

Hadar, I., & Leron, U. (2008). How intuitive is object-oriented design. *Communications of the ACM, 51*(5), 41-46. Retrieved October 20, 2009, from EBSCOhost database.

Hadar, I., Sherman, S., & Hazzan, O. (2008). Learning human aspects of collaborative software development. *Journal of Information Systems Education*, *19*(3), 311-319. Retrieved July 28, 2009, from EBSCOhost database.

Harris, M., Aebischer, K., & Klaus, T. (2007). The whitewater process: Software product development in small IT business. *Communications of the ACM, 50*(5), 89-93. Retrieved August 25, 2009, from EBSCOhost database.

Hashmi, S. I., & Baik, J. (2007). Software quality assurance in XP and spiral - A comparative study. *Fifth International Conference on Computational Science and Applications*, 367-374. doi: 10.1109/ICCSA.2007.65

Haustein, S., & Pleumann, J. (2005). A model-driven runtime environment for web applications. *Software and Systems Modeling*, *4*(4), 443-458. doi: 10.1007/s10270-005-0093-2

Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J. P., & Margaria, T. (2008). Software engineering and formal methods. *Communications of the ACM*, *51*(9), 54-59. doi: 10.1145/1378727.1378742

Horn, C. (2009). Why has software been so difficult to write well? *Engineers Journal, 63*(9), 476-479. Retrieved January 22, 2010, from EBSCOhost database.

Iacob, I. M., & Constantinescu, R. (2008). Testing. First step towards software quality. *Journal of Applied Quantitative Methods, 3*(3), 241-253. Retrieved July 1, 2009 from EBSCOhost database.

Ilies, R., Judge, T., & Wagner, D. (2006). Making sense of motivational leadership: The trail from transformational leaders to motivated followers. *Journal of Leadership and Organizational Studies*, *13*(1), 1-22. Retrieved September 6, 2009, from EBSCOhost database.

Institute of Electrical and Electronics Engineers. (2004). *Guide to the software engineering body of knowledge*. Los Alamitos, CA: Author.

International Organization for Standardization. (2004a). *ISO/IEC 15504-2:2004: Information technology - process assessment - part 2: performing an assessment*. (ISO/IEC 15504-2:2004). Geneva, Switzerland: Author.

International Organization for Standardization. (2004b). *ISO/IEC 15504-1:2004 Information technology process assessment part 1: concepts and vocabulary*. (ISO/IEC 15504-1:2004). Geneva, Switzerland: Author.

Jain, A. (2007). A value-based theory of software engineering. (Ph. D. dissertation, University of Southern California, 2007). Retrieved March 6, 2010, from ProQuest Dissertations and Theses: Full Text. (Publication No. AAT3311031).

Jakimi, A., & Elkoutbi, M. (2009). A new approach for UML scenario engineering. *International Review on Computers and Software*, *4*(1), 88-95. Retrieved August 28, 2009, from EBSCOhost database.

Jianguo, L., Jinghui, L., & Hongbo, L. (2008). Research on software process improvement model based on CMM. *Proceedings of World Academy of Science: Engineering and Technology*, *29*, 368-371. Retrieved October 20, 2008, from EBSCOhost database.

Johnson, J., Boucher, K. D., Connors, K., & Robinson, J. (2001). Collaboration: Development and management collaborating on project success. *The Software Magazine*. Retrieved from http://www.softwaremag.com/archive/2001feb/CollaborativeMgt-II.html

Johnson, M. (2008). Multi-project staffing: An agile based framework. (D.P.S. dissertation, Pace University, 2008). Retrieved March 6, 2010, from ProQuest Dissertations and Theses: Full Text. (Publication No. AAT3337583).

Johnson-Cramer, M. E., Parise, S., & Cross, R. L. (2007). Managing change through networks and values. *California Management Review*, *49*(3), 85-109. Retrieved June 12, 2008, from EBSCOhost database.

Jones, C. (2008). Geriatric issues of aging software. *Journal of the Quality Assurance Institute*, *22*(2), 21-27. Retrieved August 7, 2009, from EBSCOhost database.

Keller, R. D., Marose, R. A., & Schuessler, T. (2009). Six sigma project - job tracking system. *Proceedings for the Northeast Region Decision Sciences Institute (NEDSI)*, 437-442. Retrieved August 31, 2009, from EBSCOhost database.

Kendall, R., Fisher, D., Henderson, D., Carver, J. C., Mark, A., Post, D., et al. (2008). Development of a weather forecasting code: A case study. *IEEE Software*, *25*(4), 59-65. Retrieved August 20, 2009, from ProQuest database.

Kenett, R. S., & Baker, E. R. (2010). *Process improvement and CMMI for systems and software.* Boca Raton, FL: Taylor and Francis Group, LLC.

Kerzner, H. (2009). *Project management: A systems approach to planning, scheduling, and controlling* (10th ed.). Hoboken, NJ: John Wiley & Sons, Inc.

Keston, G. (2008). *Agile software development* (00011494). Faulkner Information Services. Retrieved from Faulkner's Advisory on Computer and Communications Technologies

King, D. R. (2007). Balanced innovation management. *Defense Acquisition Review Journal*, 150-169.

Kirova, V., Kirby, N., Kothari, D., & Childress, G. (2008). Effective requirements traceability: Models, tools, and practices. *Bell Labs Technical Journal, 12*(4), 143-158. Retrieved August, 20, 2009, from EBSCOhost database.

Kneuper, R. (2009). *CMMI: Improving software and system development processes using capability maturity model integration (CMMI-DEV).* Santa Barbara, CA: Rocky Nook Inc.

Kotlarsky, J., Oshri, H., & Willcocks, L. (2007). Social ties in globally distributed software teams: beyond face-to-face meetings. *Journal of Global Information Technology Management*, *10*(4), 7-34. Retrieved September 3, 2009, from EBSCOhost database.

Kruchten, P. (2008). Licensing software engineers. *IEEE Software*, *25*(6), 35-37. Retrieved August 8, 2009, from ProQuest database.

Landaeta, R. E. (2008). Evaluating benefits and challenges of knowledge transfer across project. *Engineering Management Journal*, *20*(1), 29-38. Retrieved July 4, 2009, from EBSCOhost database.

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, *36*(6), 47-56. Retrieved August 26, 2009, from IEEE Computer Society Digital Library database.

Lavrishcheva, E. M. (2008). Software engineering as a scientific and engineering discipline. *Cybernetics and Systems Analysis*, *44*(3), 324-332. Retrieved November 10, 2008, from ProQuest database.

Lee, G., Delone, W., & Espinosa, J. A. (2006). Ambidextrous coping strategies in globally distributed software development projects. *Communications of the ACM*, *49*(10), 35-40. Retrieved September 3, 2009, from EBSCOhost database.

Leedy, P. D., & Ormrod, J. E. (2005). *Practical research: Planning and design* (8th ed.). Upper Saddle River, NJ: Pearson Education, Inc.

Lewis, L. K. (2006). Employee perspectives on implementation communication as predictors of perceptions of success and resistance. *Western Journal of Communication*, *70*(1), 23-46. doi: 10.1080/10570310500506631

Li, E. Y., Chen, H., & Lee, T. (2003). A longitudinal study of software process management in Taiwan's top companies. *Total Quality Management, 14*(5), 571-590. doi: 10.1080/1478336032000053591

Lindbom, D. (2007). A culture of coaching: The challenge of managing performance for long-term results. *Organization Development Journal*, *25*(2), 101-106. Retrieved December 12, 2008, from EBSCOhost database.

Linden, G., Ortega, R., & Hong, J. (2010). Software engineering, smartphones and health systems, and security warnings. *Communications of the ACM, 53*(1), 16-17. doi: 10.1145/1629175.1629181

Mahoney, M. S. (2008). What makes the history of software hard? *IEEE Annals of the History of Computing*, *30*(3), 8-18. Retrieved August 22, 2009, from IEEE Computer Society Digital Library database.

Masood, S. A., Dani, S. S., Burns, N. D., & Backhouse, D. J. (2006). Transformational leadership and organizational culture: The situational strength perspective. *Proceedings of the Institution of Mechanical Engineers - Part B - Engineering Manufacture*, *220*(6), 941-949. doi: 10.1243/09544054JEM499

Mathew, E. G. (2008). Knowledge management progression, issues, and approaches for organizational effectiveness in manufacturing industry: An implementation agenda. *ICFAI Journal of Knowledge Management, 6*(1), 20-45. Retrieved May 6, 2009, from EBSCOhost database.

Mathew, V., & Kavitha, M. (2008). The critical knowledge transfer in an organization: Approaches. *ICFAI Journal of Knowledge Management*, *6*(4), 25-39. Retrieved August 15, 2009, from EBSCOhost database.

McCleaf, K. J. (2007). Sexual minority women, identity development, and acquisition of academic success: A qualitative study. (Doctoral dissertation, University of Phoenix, 2007). Retrieved January 21, 2010, from ProQuest Dissertations and Theses @ University of Phoenix. (Publication No. AAT 3299235).

McManus, J., & Wood-Harper, T. (2007a). Software engineering: A quality management perspective. *The TQM Magazine*, *19*(4), 315-327. doi: 10.1108/09544780710756223

McManus, J., & Wood-Harper, T. (2007b). Understanding the sources of information systems project failure. *Management Services*, *51*(3), 38-43. Retrieved July 29, 2009, from EBSCOhost database.

McMillan, J. H., & Schumacher, S. (2006). *Research in education: Evidence-based inquiry* (6th ed.). New York: Pearson Education, Inc.

Miller, D., & Desmarais, S. (2007). Developing your talent to the next level: Five best practices for leadership development. *Organization Development Journal*, *25*(3), 37-43. Retrieved December 12, 2007, from EBSCOhost database.

Miller, W. L. (2006). Innovation rules. *Research Technology Management*, *49*(2), 8-14. Retrieved August 18, 2009, from ProQuest database.

Mishra, D., & Mishra, A. (2008). Effective software review process for small and medium enterprises. *IET Software, 1*(4), 132-142. Retrieved October 21, 2009, from EBSCOhost database.

Mizell, C., & Malone, L. (2007). A project management approach to using simulation for cost estimation on large, complex software development projects. *Engineering Management Journal, 19*(4), 30-36. Retrieved November 10, 2009, from ProQuest database.

Mohtashami, M., Marlowe, T., Kirova, V., & Deek, F. P. (2006). Risk management for collaborative software development. *Information Systems Management*, *23*(4), 20-30. Retrieved November 20, 2008, from ProQuest database.

Monalisa, M., Daim, T., Mirani, F., Dash, P., Khamis, R., & Bhusari, V. (2008). Managing global design teams. *Research Technology Management, 51*(4), 48-59. Retrieved August 21, 2009, from EBSCOhost database.

Mukherjee, I. (2008). Understanding information system failures from the complexity perspective. *Journal of Social Sciences, 4*(4), 308-319. Retrieved January 22, 2010, from EBSCOhost database.

Murray T. M. (2008). A grounded theory of U.S. Army installation realignment and closure leadership characteristics. (Doctoral dissertation, University of Phoenix, 2008). Retrieved March 9, 2010, from ProQuest Dissertations and Theses @ University of Phoenix. (Publication No. AAT 3326209).

Nasution, M. F., & Weistroffer, H. R. (2009). Documentation in systems development: A significant criterion for project success. *42nd Hawaii International Conference on System Sciences*, 1-9. Retrieved August 26, 2009, from IEEE Computer Society Digital Library database.

National Defense University. (2009). *Strategic leadership and decision making.* Retrieved August 14, 2009, from http://www.au.af.mil/au/awc/awcgate/ndu/strat-ldr-dm/cont.html

Neuman, W. L. (2003). *Social research methods: Qualitative and quantitative approaches* (5th ed.). New York: Pearson Education Inc.

Neumann, P. G. (2008). Reflections on computer-related risks. *Communications of the ACM*, *51*(1), 78-80. Retrieved February 19, 2009, from EBSCOhost database.

Nevo, D., & Wade, M. R. (2007). How to avoid disappointment by design. *Communications of the ACM*, *50*(4), 43-48. Retrieved August 20, 2009, from EBSCOhost database.

Nielsen, M. F. (2009). Interpretative management in business meetings. *Journal of Business Communication*, *46*(1), 23-56. doi: 10.1177/0021943608325752

Northouse, P. G. (2010). *Leadership: Theory and practice* (5th ed.). Thousand Oaks, CA: Sage Publications, Inc.

Ogle, A. (2009). Making sense of the hotel guestroom. *Journal of Retail and Leisure Property, 8*(3), 159-172. Retrieved January 21, 2010, from EBSCOhost database.

Peppard, J., Ward, J., & Daniel, E. (2007). Managing the realization of business benefits from IT investments. *MIS Quarterly Executive*, *6*(1), 1-11. Retrieved February 22, 2009, from EBSCOhost database.

Persichitte, K. A., Young, S., & Tharp, D. D. (1997). *Conducting research on the Internet: Strategies for electronic interviewing.* Retrieved January 21, 2010, from ERIC Document Reproduction Service No. ED409860.

Persse, J. R. (2006). *Process improvement essentials: CMMI, ISO 9001, six sigma.* Sebastopol, CA: O'Reilly Media Inc.

Peslak, A. R., Subramanian, G. H., & Clayton, G. E. (2008). The phases of ERP software implementation and maintenance: A model for predicting preferred ERP use. *Journal of Computer Information Systems*, *48*(2), 25-33. Retrieved August 17, 2009, from EBSCOhost database.

Pfleeger, S. L., & Atlee, J. M. (2010). *Software engineering: Theory and practice* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Pino, F., Garcia, F., & Piattini, M. (2008). Software process improvement in small and medium software enterprises: A systematic review. *Software Quality Journal*, *16*(2), 237-252. doi: 10.1007/s11219-007-9038-z

Polkinghorne, D. E. (2005). Language and meaning: Data collection in qualitative research. *Journal of Counseling Psychology, 52*(2), 127-145. doi: 10.1037/0022-0167.52.2.137

Pozgaj, Q., Sertic, H., & Boban, M. (2007). Effective information systems development as a key to successful enterprise. *Management*, *12*(1), 65-86. Retrieved December 7, 2008, from EBSCOhost database.

Pressman, R. S. (2010). *Software engineering a practitioner's approach* (7th ed.). New York: McGraw-Hill.

Probert, D., Hunt, F., Fraser, P., Fleury, A., & Holden, T. (2007). Sourcing software content for manufactured products. *Proceedings of the Institution of Mechanical Engineers -- Part B -- Engineering Manufacture*, *221*(5), 809-820. doi: 10.1243/09544054JEM532

QSR International. (2008). *NVivo8.* Retrieved October 31, 2010, from http://www.qsrinternational.com/products_previous-products.aspx

Rajlich, V. (2006). Changing the paradigm of software engineering. *Communications of the ACM, 49*(8), 67-70. Retrieved January 24, 2010, from EBSCOhost database.

Ramasubbu, N., Mithas, S., Krishan, M. S., & Kemerer, C. F. (2008). Work dispersion, process-based learning, and offshore software development performance. *MIS Quarterly*, *32*(2), 437-458. Retrieved September 25, 2009, from EBSCOhost database.

Ramesh, B., Cao, L., Mohan, K., & Xu, P. (2006). Can distributed software development be agile? *Communications of the ACM*, *49*(40), 41-46. Retrieved September 3, 2009, from EBSCOhost database.

Rasulzadeh, S. (2008). Formal modeling and verification of software models. *Proceedings of World Academy of Science, Engineering and Technology*, *32*, 276-282. Retrieved August 17, 2009, from EBSCOhost database.

Rubinstein, D. (2007, March 1). Standish group report: There's less development chaos today. *Software Development Times*. Retrieved from http://www.sdtimes.com/link/30247

Salinas, M. P. C., Prudhomme, G., & Brissaud, D. (2008). Requirement-oriented activities in an engineering design process. *International Journal of Computer Integrated Manufacturing, 21*(2), 127-138. doi: 10.1080/09511920701607816

Salkind, N. J. (2003). *Exploring research* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Sapienza, A. M. (2005). From the inside: Scientists' own experience of good (and bad) management. *Research and Development Management*, *35*(5), 473-482. doi: 10.1111/j.1467-9310.2005.00404.x

Schneidewind, N. (2007). A quantitative approach to software development using IEEE 982.1 [Electronic version]. *IEEE Software*, *24*(1), 65-72. Retrieved July 29, 2009, from ProQuest database.

Schram, T. H. (2006). *Conceptualizing and proposing qualitative research* (2nd ed.). Upper Saddle River, NJ: Merrill Prentice Hall.

Scott, W. R., & Davis, G. F. (2007). *Organizations and organizing: Rational, natural, and open systems perspectives*. Upper Saddle River, NJ: Prentice Hall.

Sen, Z., & Zheng, Y. (2007). The relation of CMM and software lifecycle model. *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, *3*, 864-869. doi: 10.1109/SNPD.2007.318

Shank, G. D. (2006). *Qualitative research: A personal skills approach* (2nd ed.). Upper Saddle River, NJ: Merrill Prentice Hall.

Shenvi, A. A. (2008). Design for six sigma: Software product quality. *Proceedings of the 1st Conference on India Software Engineering*, 97-106. Retrieved August 31, 2009, from ACM Digital Library database.

Siddiqui, M. S., Hussain, S. J., & Hussain, S. (2006). Comprehensive software development model. *IEEE International Conference on Computer Systems and Applications*, 353-360. Retrieved August 26, 2009, from IEEE Computer Society Digital Library database.

Sommerville, I. (2007). *Software engineering* (8th ed.). Harlow, England: Pearson Education Limited.

Stackpole, B. (2008). All systems go. *Design News*, *63*(5), 61-63. Retrieved August 5, 2009, from EBSCOhost database.

Steeneken, K. (2009). Towards an integral leadership vision in software development. *Integral Leadership Review,* 1-11. Retrieved January 24, 2010, from EBSCOhost database.

Stephenson, S. V., & Sage, A. P. (2007). Information and knowledge perspectives in systems engineering and management for innovation and productivity through enterprise resource planning. *Information Resources Management Journal*, *20*(2), 44-61. Retrieved November 10, 2008, from ProQuest database.

Subramanian, A. M., & Soh, P. (2008). Knowledge integration and effectiveness of open source software development projects. *IIMB Management Review*, *20*(2), 139-148. Retrieved August 19, 2009, from EBSCOhost database.

Subramanian, G. H., Klein, G., Jiang, J. J., & Chan, C. (2009). Balancing four factors in system development projects. *Communications of the ACM*, *52*(10), 118-121. doi: 11.1145/1562764.1562794

Sun, Y. (2008). Business-oriented software process improvement based on CMM and CMMI using QFD. (Ph. D. dissertation, Missouri University of Science and Technology, 2008). Retrieved March 6, 2010, from ProQuest Dissertations and Theses: Full Text. (Publication No. AAT3318731).

Suzuki, L. A., Ahluwalia, M. K., Arora, A. K., & Mattis, J. A. (2007). The pond you fish in determines the fish you catch: Exploring strategies for qualitative data collection. *The Counseling Psychologist*, *35*(2), 295-327. doi: 10.1177/0011000006290983

Tarabishy, A., Solomon, G., Fernald, L. W., & Sashkin, M. (2005). The entrepreneurial leader's impact on organization's performance in dynamic markets. *Journal of Private Equity*, *8*(4), 20-29. Retrieved September 7, 2009, from EBSCOhost database.

Taylor, V. (2007). Leadership for service improvement: part 3. *Nursing Management*, *14*(1), 28-32. Retrieved August 9, 2009, from EBSCOhost database.

Telang, R., & Wattal, S. (2007). An empirical analysis of the impact of software vulnerability announcements on firm stock. *IEEE Transactions on Software Engineering*, *33*(8), 544-562. doi: 10.1109/TSE.2007.1015

Tesch, D., Kloppenborg, T., & Frolick, M. (2007). IT project risk factors: The project management professional perspective. *The Journal of Computer Information Systems*, *47*(4), 61-70. Retrieved October 4, 2008, from ProQuest database.

The Standish Group. (2009). *CHAOS summary 2009*. [Brochure]. New Yamouth, Massachusetts: Author.

Wallace, L., & Keil, M. (2004). Software projects risks and their effect on outcomes. *Communications of the ACM*, *47*(4), 68-73. Retrieved October 3, 2008, from EBSCOhost database.

Warzynski, C. C. (2005). The evolution of organizational development at Cornell University: Strategies for improving performance and building capacity. *Advances in Developing Human Resources*, *7*(3), 338-352. doi: 10.1177/1523422305277175

Wirfs-Brock, R. J. (2008). Designing then and now. *IEEE Software*, *25*(6), 29-31. Retrieved July 20, 2009, from ProQuest database.

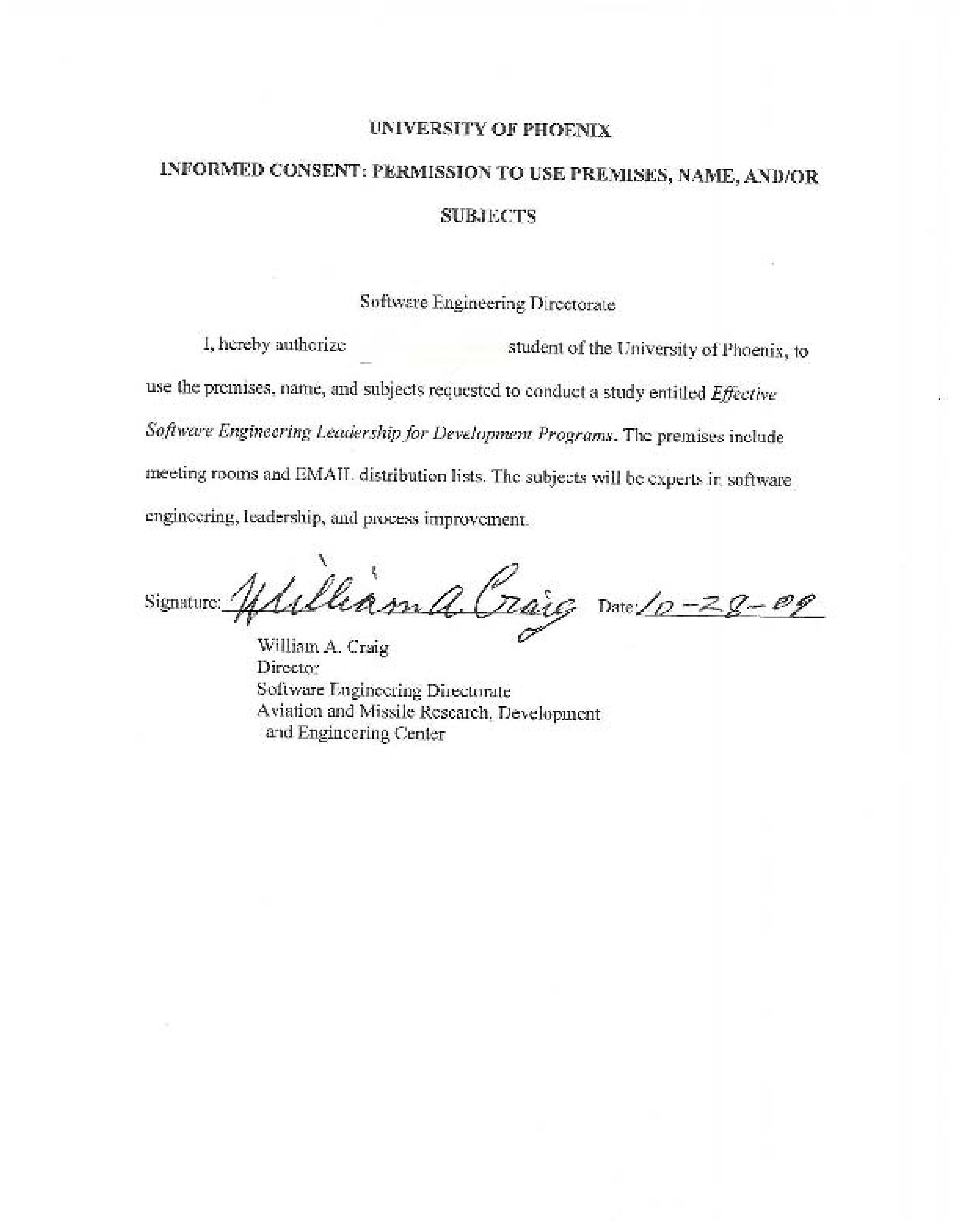
Wirth, N. (2008). A brief history of software engineering. *IEEE Annals of the History of Computing*, *30*(3), 32-39. Retrieved August 20, 2009, from IEEE Computer Society Digital Library database.

Woolridge, R. W., Hale, D. P., Hale, J. E., & Sharpe, R. S. (2009). Software project scope alignment: An outcome-based approach. *Communications of the ACM*, *52*(7), 147-152. doi: 10.1145/1538788.1538822

Xu, L., & Brinkkemper, S. (2007). Concepts of product software. *European Journal of Information Systems*, *16*(5), 531-542. Retrieved August 8, 2009, from ProQuest database.

Yang, F., & Mei, H. (2006). Development of software engineering: Co-operative efforts from academia, government and industry. *Proceedings of the 28th International Conference on Software Engineering*, 2-9. Retrieved August 21, 2009, from ACM Digital Library database.

APPENDIX A: PERMISSION TO USE PREMISES



UNIVERSITY OF PHOENIX

INFORMED CONSENT: PERMISSION TO USE PREMISES, NAME, AND/OR SUBJECTS

Software Engineering Directorate

I, hereby authorize   student of the University of Phoenix, to use the premises, name, and subjects requested to conduct a study entitled *Effective Software Engineering Leadership for Development Programs*. The premises include meeting rooms and EMAIL distribution lists. The subjects will be experts in software engineering, leadership, and process improvement.

Signature: William A. Craig Date: 10-28-09

William A. Craig
Director
Software Engineering Directorate
Aviation and Missile Research, Development
and Engineering Center

APPENDIX B: STUDY INVITATION

UNIVERSITY OF PHOENIX

INFORMED CONSENT: PARTICIPANTS 18 YEARS OF AGE AND OLDER

Dear Participant,

My name is name of student and I am a student at the University of Phoenix working on a Doctor of Management in Organizational Leadership for Information System Technology degree. I am conducting a research study entitled Effective Software Engineering Leadership for Development Programs. The purpose of the research study is to investigate leadership practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes.

I would like to invite you to participate in the research study. If you decide to participate, the study will require approximately 20 minutes to complete an electronic questionnaire on software engineering. Your participation in this study is voluntary. If you choose not to participate or to withdraw from the study at any time, you can do so without penalty or loss of benefit to yourself. The results of the research study may be published but your identity will remain confidential and your name will not be disclosed to any outside party.

In this research, there are no foreseeable risks to you.

Although there may be no direct benefit to you, a possible benefit of your participation is the feedback provided could result in the identification of new leadership processes, approaches, and methodologies for improving software development initiatives.

If you have any questions concerning the research study, please call me at student phone number or NAME@email.phoenix.edu

As a participant in this study, you should understand the following:

1. You may decline to participate or withdraw from participation at any time without consequences.
2. Your identity will be kept confidential.
3. The researcher has thoroughly explained the parameters of the research study and all of your questions and concerns have been addressed.
4. Data will be stored in a secure and locked area. The data will be held for a period of 3 years, and then destroyed.
5. The research results may be used for publication.

SURVEY IS AVAILABLE AT:

APPENDIX C: INFORMED CONSENT

UNIVERSITY OF PHOENIX

Informed Consent: Participants 18 years of age and older

Dear Participant,

My name is name of student and I am a student at the University of Phoenix working on a Doctor of Management in Organizational Leadership for Information System Technology degree. I am conducting a research study entitled Effective Software Engineering Leadership for Development Programs. The purpose of the research study is to investigate leadership practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes.

Your participation will involve completing an online questionnaire requiring approximately 20 minutes. Your participation in this study is voluntary. If you choose not to participate or to withdraw from the study at any time, you can do so without penalty or loss of benefit to yourself. The results of the research study may be published but your identity will remain confidential and your name will not be disclosed to any outside party.

In this research, there are no foreseeable risks to you.

Although there may be no direct benefit to you, a possible benefit of your participation is the feedback provided could result in the identification of new leadership processes, approaches, and methodologies for improving software development initiatives.

If you have any questions concerning the research study, please call me at student phone number or NAME@email.phoenix.edu

As a participant in this study, you should understand the following:

1. You may decline to participate or withdraw from participation at any time without consequences.
2. Your identity will be kept confidential.
3. The researcher has thoroughly explained the parameters of the research study and all of your questions and concerns have been addressed.
4. Data will be stored in a secure and locked area. The data will be held for a period of 3 years, and then destroyed.
5. The research results may be used for publication.

By electronically accepting the study conditions, you acknowledge that you understand the nature of the study, the potential risks to you as a participant, and the means by which your identity will be kept confidential. Your electronic acceptance on this form also indicates that you are 18 years old or older and that you give your permission to voluntarily serve as a participant in the study described.

CONTINUE – I agree to participate in the study

EXIT – I do not want to participate in the study

APPENDIX D: QUESTIONNAIRE

Software Leadership Interview

Section 1 – INFORMED CONSENT

UNIVERSITY OF PHOENIX

Informed Consent: Participants 18 years of age and older

Dear Participant,

My name is name of student and I am a student at the University of Phoenix working on a Doctor of Management in Organizational Leadership for Information System Technology degree. I am conducting a research study entitled *Effective Software Engineering Leadership for Development Programs*. The purpose of the research study is to investigate leadership practices applied to software development programs to determine which processes are effective, beneficial, and applicable to achieving successful program outcomes.

Your participation will involve completing an online interview questionnaire requiring approximately 20 minutes. Your participation in this study is voluntary. If you choose not to participate or to withdraw from the study at any time, you can do so without penalty or loss of benefit to yourself. The results of the research study may be published but your identity will remain confidential and your name will not be disclosed to any outside party.

In this research, there are no foreseeable risks to you.

Although there may be no direct benefit to you, a possible benefit of your participation is the feedback provided could result in the identification of new leadership processes, approaches, and methodologies for improving software development initiatives.

If you have any questions concerning this research study, please contact me at student phone number or NAME@email.phoenix.edu

As a participant in this study, you should understand the following:

1. You may decline to participate or withdraw from participation at any time without consequences.
2. Your identity will be kept confidential.
3. The researcher has thoroughly explained the parameters of the research study and all of your questions and concerns have been addressed.
4. Data will be stored in a secure and locked area. The data will be held for a period of 3 years, and then destroyed.
5. The research results may be used for publications.

By electronically accepting the study conditions, you acknowledge that you understand the nature of the study, the potential risks to you as a participant, and the means by which your identity will be kept confidential. Your electronic acceptance on this form also indicates that you are 18 years old or older and that you give your permission to voluntarily serve as a participant in the study described.

_____ CONTINUE – I agree to participate in the study

_____ EXIT – I do not want to participate in the study

Section 2 – Background Information

1. Which title best describes your job function?

_____ Executive Leadership
_____ Team Leader
_____ Senior Software/System Engineer
_____ Software/System Engineer
_____ Team Member
_____ Administrative Support
_____ If other, please specify

2. How many years experience do you have leading software development efforts?

_____ Less than 1 year
_____ 1 – 5 years
_____ 6 – 10 years
_____ 11 – 15 years
_____ 16 – 20 years
_____ More than 20 years

3. How many years of experience do you have in software engineering?

_____ Less than 1 year
_____ 1 – 5 years
_____ 6 – 10 years
_____ 11 – 15 years
_____ 16 – 20 years
_____ More than 20 years

Section 3 – Leadership and Software Process Questionnaire

4. What do you feel are the most effective leadership approaches for successful programs? Why do you feel these approaches are effective?

5. What do you feel is the most important leadership characteristic to support software development programs? Why do you feel this characteristic is the most important?

6. In your experience, what leadership capabilities or approaches have contributed to the success of software development programs? Why do you feel they contributed to program success?

7. In your experience, what leadership capabilities or approaches have contributed to the delay or failure of software development programs? Why do you feel they contributed to program delay or failure?

8. What do you feel is the most important process for successful software development? Why do you feel this is the most important process?

9. What software development methodologies have you applied that are the most effective? Why do you feel these methodologies were the most effective?

10. What software development methodologies have you applied that are ineffective? Why do you feel these methodologies were ineffective?

11. Please provide any additional comments or insights you would like to share on software leadership capabilities or software processes.

Section 4 – Consent to Survey

Thank you for participating in the study.

Do you consent to including your responses in the research study?

______ YES – Include my responses to the study

______ NO – Withdraw from the study and delete all responses

APPENDIX E: DEMOGRAPHIC INFORMATION FOR OPEN-ENDED QUESTIONS

Table E-1

Job Functions for Respondents to Open-Ended Questions

	Open-Ended Question							
	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Total Respondents	68	69	57	61	54	50	46	37
Job Function								
Executive Leadership	4	4	4	4	4	3	2	2
Team Leader	18	19	17	18	14	15	14	10
Senior Software/ System Engineer	20	20	12	15	13	11	10	9
Software/ System Engineer	16	16	16	16	16	14	14	11
Team Member	5	5	4	4	4	4	3	2
Administrative Support	1	1	1	1	0	0	0	1
Other	4	4	3	3	3	3	3	2

Table E-2

Experience Leading Software Development for Respondents to Open-Ended Questions

	Open-Ended Question							
	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Total Respondents	68	69	57	61	54	50	46	37
Years of Experience Leading Software Development								
Less Than 1 Year	11	11	9	9	7	7	7	5
1 – 5 Years	16	17	14	17	14	12	10	10
6 – 10 Years	16	16	12	13	13	12	12	8
11 – 15 Years	10	10	9	9	9	9	8	7
16 – 20 Years	9	9	7	7	6	6	5	4
More Than 20 Years	6	6	6	6	5	4	4	3

Table E-3

Experience in Software Engineering for Respondents to Open-Ended Questions

	Open-Ended Question							
	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Total Respondents	68	69	57	61	54	50	46	37
Years of Experience in Software Engineering								
Less Than 1 Year	7	7	5	5	4	4	4	3
1 – 5 Years	13	13	12	13	12	11	11	9
6 – 10 Years	12	13	11	11	9	9	8	6
11 – 15 Years	8	8	7	8	6	6	5	4
16 – 20 Years	11	11	7	9	9	8	7	5
More Than 20 Years	17	17	15	15	14	12	11	10