2021-12

USEFUL MEASURES OF COMPLEXITY: A MODEL OF ASSESSING DEGREE OF COMPLEXITY IN ENGINEERED SYSTEMS AND ENGINEERING PROJECTS

Ton, Cuong

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

USEFUL MEASURES OF COMPLEXITY: A MODEL OF ASSESSING DEGREE OF COMPLEXITY IN ENGINEERED SYSTEMS AND ENGINEERING PROJECTS

by

Cuong Ton

December 2021

Dissertation Supervisors: Ronald E. Giachetti
Robert C. Harney

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2021	**3. REPORT TYPE AND DATES COVERED** Dissertation

4. TITLE AND SUBTITLE USEFUL MEASURES OF COMPLEXITY: A MODEL OF ASSESSING DEGREE OF COMPLEXITY IN ENGINEERED SYSTEMS AND ENGINEERING PROJECTS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Cuong Ton	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Many modern systems are very complex, a reality which can affect their safety and reliability of operations. Systems engineers need new ways to measure problem complexity. This research lays the groundwork for measuring the complexity of systems engineering (SE) projects. This research proposes a project complexity measurement model (PCMM) and associated methods to measure complexity. To develop the PCMM, we analyze four major types of complexity (structural complexity, temporal complexity, organizational complexity, and technological complexity) and define a set of complexity metrics.

Through a survey of engineering projects, we also develop project profiles for three types of software projects typically used in the U.S. Navy to provide empirical evidence for the PCMM. The results of our work on these projects show that as a project increases in complexity, the more difficult and expensive it is for a project to meet all requirements and schedules because of changing interactions and dynamics among the project participants and stakeholders. The three projects reveal reduction of project complexity by setting a priority and a baseline in requirements and project scope, concentrating on the expected deliverable, strengthening familiarity of the systems engineering process, eliminating redundant processes, and clarifying organizational roles and decision-making processes to best serve the project teams while also streamlining on business processes and information systems.

14. SUBJECT TERMS project complexity, project risk, complexity science, complex systems, systems engineering, SE complex systems engineering, complexity reduction, measurement, measures of complexity, model of estimating degree of complexity, approaches to measure system complexity, methods to estimate the complexity of engineered systems, interdependence, complexity profile, project complexity measurement model, PCMM	**15. NUMBER OF PAGES** 357
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

USEFUL MEASURES OF COMPLEXITY: A MODEL OF ASSESSING DEGREE OF COMPLEXITY IN ENGINEERED SYSTEMS AND ENGINEERING PROJECTS

Cuong Ton
Civilian, Department of the Navy
BSEE, University of Utah, 1987
MEng, University of Utah, 1989
MS, Electrical Engineering, Naval Postgraduate School, 2013

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2021

Approved by:

Ronald E. Giachetti
Department of
Systems Engineering
Dissertation Supervisor

Robert C. Harney (posthumously)
Department of
Systems Engineering
Dissertation Supervisor

Wei Kang
Department of
Applied Mathematics

Robert Semmens
Department of
Systems Engineering

Douglas L. Van Bossuyt
Department of
Systems Engineering

Clifford A. Whitcomb
Department of
Systems Engineering

Ronald E. Giachetti
Department of
Systems Engineering
Dissertation Chair

Approved by:

Oleg A. Yakimenko
Chair, Department of Systems Engineering

Michael E. Freeman
Vice Provost of Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Many modern systems are very complex, a reality which can affect their safety and reliability of operations. Systems engineers need new ways to measure problem complexity. This research lays the groundwork for measuring the complexity of systems engineering (SE) projects. This research proposes a project complexity measurement model (PCMM) and associated methods to measure complexity. To develop the PCMM, we analyze four major types of complexity (structural complexity, temporal complexity, organizational complexity, and technological complexity) and define a set of complexity metrics.

Through a survey of engineering projects, we also develop project profiles for three types of software projects typically used in the U.S. Navy to provide empirical evidence for the PCMM. The results of our work on these projects show that as a project increases in complexity, the more difficult and expensive it is for a project to meet all requirements and schedules because of changing interactions and dynamics among the project participants and stakeholders. The three projects reveal reduction of project complexity by setting a priority and a baseline in requirements and project scope, concentrating on the expected deliverable, strengthening familiarity of the systems engineering process, eliminating redundant processes, and clarifying organizational roles and decision-making processes to best serve the project teams while also streamlining on business processes and information systems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CBO	coupling between object classes
CCB	configuration control board
CMMI	capability maturity model integration
COSYSMO	constructive systems engineering cost model
CSE	complex systems engineering
CSM	complexity study method
DARPA	Defense Advanced Research Projects Agency
DIT	depth of inheritance tree
IDD	interface design description
INCOSE	International Council on Systems Engineering
IOT	internet of things
IPMA	International Project Management Association
IPT	integrated product teams
IT	information technology
KLOC	thousand lines of code
LCOM	lack of cohesion method
LOC	lines of code
OU	organizational unit
PCMM	project complexity measurement model
PMA	program management activity
RFC	response for class
RFP	request for proposal
RTR	requirements traceability report
SARS	Severe Acute Respiratory Syndrome
SDD	software design document
SE	systems engineering
SoI	system-of-interest

SoS	system-of-systems
TFS	Team Foundation Server
TSE	traditional systems engineering
WMC	weighted methods per class

LIST OF SYMBOLS

A	a calibration constant derived from historical project data
BR	the baseline of the software release
CR_i	the number of change requirements during the i^{th} period
DDM	*d*efect density measure
E	represents economy/diseconomy of scale or an efficiency factor
EM_i	effort multiplier of COSYSMO for the ith cost driver
F	the total number of communications between two nodes *i* and *j*
f_{ij}	the frequency of communications between two nodes *i* and *j*
GD	measure of geographical distribution of teams
IF	index function
JPT	Number of different job position types
M	mass
n	the number of organizations involved in the communications
P	power
PM	effort in person-month for nominal schedule
PS_i	the project size at the i^{th} period
RVM	requirements volatility measure
$Size$	the weighted sum of the four size drivers (requirements, interfaces, algorithms, operational scenarios)
ν(G)	cyclomatic complexity function
$w_{i,j}$	a factor of importance of the communication between two nodes *i* and j
WAD_i	the weighted average number of defects during the i^{th} period

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Systems engineers need to measure and reduce problem complexity to improve system affordability and maintainability. This research seeks to determine practical and effective complexity measures of engineering projects and engineered systems and to develop a measurement scheme for assessing the complexity of system development projects. For this work, we have reviewed the following relevant topics: complexity theory, attributes and characteristics of complexity, measures of complexity, project assessment, system complexity, and impacts of system complexity and project complexity.

Previous works have shown three observations. First, the concept of complexity is broad. There are no widely agreed upon complexity measurement standards for engineered systems. Second, complexity metrics for software development projects and system development projects have many variations. Most proposed complexity metrics are incomplete, impractical to use, or useful only in limited cases. Third, there are no standardized tools or methodologies to develop complex systems. Complexity theory and complex systems engineering (CSE) practices help to provide some insights and guidance in these areas.

For this research project, we have completed five important tasks. First, we conducted a literature review to identify potentially useful measures of complexity (i.e., structural complexity, organizational complexity, temporal complexity, technological complexity, numerosity, connectedness, interdependence, and diversity). Second, we developed a project complexity measurement model (PCMM) and associated methods to measure complexity in engineering projects. Third, we demonstrated the usefulness of this new approach on three engineering projects (i.e., Navy software projects B4, B6, and B8). Fourth, we established three complexity profiles for the engineering projects using the new approach and compared the computed complexity values to those from industry common practices. This provided a baseline to analyze project complexity. Fifth, we proposed complexity reduction techniques by reducing or eliminating redundant elements within designs and processes. Systems engineers can also simplify business processes with

standard tools, standard techniques, and increased familiarity with the systems engineering processes to reduce interdependence tasks and activities among organizational units.

The results of this research show that systems engineers can use the PCMM and apply its associated methods for estimating the complexity of system development projects based on six complexity metrics (number of personnel required for the development effort, defect density, organizational complexity, geographical distribution of project teams, requirements volatility, and the number of different types of job positions). The six complexity metrics are derived from the many possible complexity dimensions that scholars have observed. These six metrics are related to the project risks (e.g., schedule delays, budget overruns, and failure to meet stakeholders' requirements). By applying the PCMM to three engineering projects, we demonstrate that the method for estimating the complexity in system development projects is useful, consistent, practical, and reliable. The results of this research also show that systems engineers can use the complexity value computed by the PCMM as an indicator of project risk. From these results estimated by the PCMM, they can study the impacts and determine which kinds of complexity to reduce to improve system affordability. In addition, they can use a complexity value determined by the PCMM as a guide for project reviews, the development of the system architecture, and comparisons of alternatives.

In all three engineering projects, structural complexity, organizational complexity, temporal complexity, and technological complexity are the four main types of complexity that cause an increase in risk of schedule delays and cost overruns to the system development project. In this research, structural complexity is defined as defect density and the number of personnel required for the engineering project. Organizational complexity is defined as a measure of the patterns of communication among organizations in terms of the frequencies and levels of importance of the communications as well as a measure of geographical distribution of teams. Temporal complexity is interpreted as a measure of requirements volatility. Technological complexity is represented as a measure of the number of different types of job positions. The interaction of the project teams, dynamic behaviors in the execution target, the political culture, and the speed of

technological change also add complexity. A certain mix of values for these project-related attributes may lead to complexity.

Practitioners need a list of experience-based techniques to work on the set of problems presented by complex systems engineering (CSE). Through a review of the literature and observation in the course of the engineering projects, we find nine heuristics that can be applied as best practices in CSE. First, program managers should identify the need for subject matter experts and recruit them as part of the team. Second, systems engineers should strive to achieve a solid base of knowledge that may be applicable to the target system. Third, systems engineers should measure system complexity in the early stages of design to gain insight of a system's complexity. Fourth, systems engineers should become familiar with the target system as a user and learn what users value in similar systems. Fifth, systems engineers should be prepared to adjust their models to accommodate new findings. Sixth, systems engineers should be wary of the working model and think of ways to disprove it. Seventh, systems engineers should keep the measurements simple, short, and easy to understand. Eighth, systems engineers should work with the end in the mind and evaluate potential benefits from the measurements. Ninth, systems engineers should ask why and how the measurements can help achieve the end goal and analyze the assumptions and limitations of these measurements.

In conclusion, the justification for the new approach to assessing the degree of complexity as presented in this research project is straightforward. Project managers and systems engineers can apply the new approach of assessing the degree of complexity outlined in this research to manage engineering projects and to keep these projects below a certain complexity threshold to reduce project risk as well as system development difficulty. In other words, the new approach has the potential to reduce problem complexity and thus to improve system affordability and maintainability.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to thank Mr. Thomas Dowd (SES, Director, NAVAIR Ranges and AVMI) for his support and funding of this research.

I want to thank my advisor, Dr. Ronald Giachetti, for providing his guidance and helping me through the dissertation process. I am exceedingly grateful for his generous support.

Heartfelt thanks to Dr. Robert C. Harney for the opportunity to work on this project. Immense appreciation goes to Dr. Harney for sharing his knowledge, ideas, and time with me, and for his dedication in helping me with this project.

Thanks to my dissertation committee (Dr. Cliff Whitcomb, Dr. Douglas Van Bossuyt, Dr. Robert Semmens, and Dr. Wei Kang). I am grateful for their support. I also want to share my gratitude with Dr. Bryan O'Halloran and Ms. Lorene Barnes for their support. In addition, I want to thank Dr. David Jenn, Mr. Elissa Carey, Mr. Michael Orton, and Ms. Delores Smikle for their support through my graduate studies.

I am thankful to Betsy, Chloe, George, and Daniel for many helpful suggestions and editorial comments on my dissertation, along with their insights and perspectives.

A special thanks to John for editorial comments in reading my dissertation, and for his acumen, viewpoint, and support.

Of course, I have to thank my family, my parents, my siblings, my uncles, and my aunts for their unceasing love, encouragement, and support.

My gratitude also extends to my ninth-grade teachers, Mr. McCoy and Mr. Zahler, my high school teachers, Mr. Lott, Mr. Smith, Mr. Reynolds, Mr. Armstrong, Mr. Gilbert, Mr. Gough, Mrs. Nelson, Mrs. Carter, Mrs. Snow, Mrs. Rawson, Mr. Schofield, Mr. Clark, Mr. Gadd, Mr. Larsen, Mr. McMillan, Mrs. Prescott, Mr. Shewell, Mr. Harmon, Mr. Standley, Mr. McOmie, Mr. Perkins, Mr. Sullivan, Mr. DeNiro, Mr. Robison, and Mrs. Mann. In addition, I would like to thank my college professors, Dr. Grow, Dr. Baird, Dr. Gandhi, Dr. Iskander, Dr. Futrell, and Dr. Miller. I also want to thank my dear friends, Dawn and Beau.

Finally, I want to thank the readers of this dissertation.

THIS PAGE INTENTIONALLY LEFT BLANK

PROLOGUE

In a corner of the National Naval Aviation Museum in Pensacola, Florida, is a peculiar airplane. It is nothing like the classic airplanes (e.g., the Sopwith Camel fighter aircraft) or the modern fighter jets (e.g., the F/A-18 Hornet flown by the Blue Angels) that fill the other areas in the display section of the hanger. The immediate front of the Northrop Grumman EA-6B Prowler airplane's cockpit has a refueling probe that is asymmetrical and appears bent to the right. The top of the airplane contains an electronic warfare transmitter antenna. The canopy has a gold tint for protection from electromagnetic interference. On the back of the aircraft, the tail has a vertical fin pod extension. Looking inside, the cockpit, it is filled with various multifunctional displays, integrated communications, flight control systems, and navigation systems. These electronics look very sophisticated. They are not like the equipment we have in today's cars. The aircraft's cockpit instruments, which consist of both hardware and software, appear very difficult to operate.

So, what is it that makes up such a complex flight system? Like weaving a tapestry, each piece of technology in the flight system folds into one another to make up the system as a whole. One can imagine the complexity of integrating all the system components. One way to understand such a complex system is to develop a set of complexity measures that describe patterns of interdependent tasks and organizational activities in systems development. The purpose of understanding the complexity is to improve the system development process and to aid the development team. To analyze complexity, we must embrace the attitude of making the complex understandable. Given the quest for simplicity that propels the scientific enterprise, we hope to build in this dissertation a useful and practical model and rely on a survey of engineering projects to convey the key ideas of complexity measurement.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> The physical sciences have alternated between revolutions driven by new ideas and explorations driven by new tools.
>
> —Freeman Dyson [1],
> "Is Science Mostly Driven by Ideas or by Tools?"

This chapter begins by presenting the driving forces behind this research and introducing the notion of complexity as applied in engineered systems and engineering projects. Next, we identify several characteristics of complex engineered systems and present the research objectives of this dissertation. The subsequent section presents the anticipated benefits and contributions of the research. Finally, we describe how the rest of dissertation is organized.

A. MOTIVATION AND BACKGROUND

Project managers and systems engineers often cite complexity is a main cause of failure or difficulty in system development [2]. They believe complexity is a major contributing factor to cost escalation and schedule delay in the development of new systems [3]. In general, we observe that the higher the system's complexity, the more difficult it is to design and develop the system. Consequently, we are interested in practical measures of complexity in engineered systems and engineering projects for several reasons.

First, complexity measures are important for economic reasons. For example, project managers and systems engineers could apply the new measures of complexity outlined in this research to design systems that are below a certain complexity threshold to reduce project risk and system development difficulty. In other words, the new measures have the potential to pinpoint complexity of a problem and thus to improve system affordability and maintainability.

Second, complexity measures are important for technological reasons. Complexity science and the study of complex systems engineering (CSE) are emergent fields as

scientists strive to stay technologically competitive and to make progress in new research areas.

Third, complexity measures are important for project management. For example, by knowing the level of complexity early in a system development project, project managers can plan development efforts and apply resources in appropriate places to minimize project risk.

Fourth, both practitioners and students could gain insights about SE and design by learning and exchanging knowledge about complexity measures. Furthermore, businesses use complexity measures to gain a deeper understanding of organizational processes, and this understanding helps create customized environments that are ergonomic, efficient, practical, and effective for today's work.

Finally, complexity measures are important for government agencies and defense contractors which must quantify project complexity in requests for proposal (RFP). For example, when evaluating the architectural concept for a military system-of-systems (SoS) in a contract proposal, decision makers could use the result of an architectural complexity measurement to determine whether the architectural design of the SoS would yield a timely and cost-effective solution.

This research addresses complexity measurement as a systems engineering (SE) tool to allow a systems engineer to assess project risk and explore alternatives early in the engineering process, achieving the most cost-efficient balance. It addresses three areas of study. First, this research addresses measures of complexity of engineering projects. Specifically, this research demonstrates how to predict the level of complexity in a given engineering project in order to (1) plan development efforts, (2) apply resources in appropriate places, and (3) measure system complexity in a practical way to gain insight into system design decisions. Second, this research creates a new model of complexity measurement and applies it to engineering projects for demonstration. A survey of engineering projects is derived from three Navy software programs (named in this dissertation as projects B4, B6, and B8). Third, this research examines CSE methods to determine and understand the impact a given complexity value is likely to have on

engineering systems. With that understanding, systems engineers can adapt their SE methods to accommodate that impact.

1. Definition

Before we present the definitions of complexity, complex systems, and project complexity used in this research, let us depict and evaluate a complex scenario so that we can understand some of the complex behaviors that may be at play in a system. Furthermore, it is essential to understand complex behavior and its implications, and to characterize complex behaviors in a system in order to adapt our SE approach adequately.

Imagine a group of more than 30 cars traveling down a street with intersections. Drivers proceed based on the traffic and weather conditions as well as drivers' moods, driving habits, and urgencies. They have no common destination. Each car travels at its own speed and optimum path, and each driver's objective is to get to his or her destination accident free and with minimum delay. This scenario is complex because the traffic situation involves many environmental variables, cars' dynamics, and human interactions. We need tools from complexity science to understand traffic and to build better streets, highways, and traffic control systems. Complexities in this scenario include drivers, pedestrians, cars and their routes, guardrails, weather conditions, and speed limit signs as well as traffic signals for the crossing traffic of both automobiles and pedestrians. All of these factors can change from second to second and evolve over time.

Now, within this scenario, consider this: Paul is a systems engineer, responsible for the traffic safety of all cars on that street, and the intersections that they cross. Given the speed limit of the street and the number of cars passing through each intersection each minute, Paul has to design a traffic control system. In doing so, he must anticipate occasional rain and thunderstorms as well as pedestrians crossing at intersections. Because the traffic problem is complex, Paul relies on technology (e.g., artificial intelligence tools such as fuzzy logic models, genetic algorithms, and artificial neural networks [4]) to build a traffic control system to solve the traffic problem. The control system might introduce additional complexity to the problem. For example, because every systems engineer has his or her unique interpretation of the causes of the traffic problem (e.g., vehicles breaking down during rush hour, vehicles driving too slow on the streets, weather, traffic volume,

or pedestrians crossing streets) and how to solve it, engineers might impose requirements to address such events in order to prevent traffic jams and collisions. As described in the traffic example, a traffic control model depends on many variables, mechanisms and parameters, and it is not so obvious which traffic control models are the most effective in preventing traffic jams and collisions. In the traffic control system development project, Paul may need to address the following two questions: (1) How many operational problems could result from the poor design? (2) How many testing issues may stem from difficult requirements?

Complexity both hinders and benefits the world in which we live. For example, in automobiles, engineers design and install an autopilot module to enable auto-park and automatic driving on highways. These features benefit the consumers, but at the same time, consumers are paying more for the automobile and potentially taking on some additional risks by using these features. This illustrates that complexity is often the price of increased performance. However, the development of self-driving cars continues around the globe. Cars without a self-driving capability contain around 100 million lines of code (LOC) [5]. Self-driving cars require about one billion LOC [6]. The F-35 fighter jets contain around 24 million LOC [7]. Evaluating by LOC, an autonomous car could be up to four times more complex than the F-35 fighter jet. In fact, self-driving cars have many engineering challenges. For companies in the competition to market the technology of self-driving car, they face a very challenging task because they must demonstrate the safety of these vehicles in a cost-effective way and within a practical timeline [8]. These companies need to perform physical testing and that would require hundreds of thousands of driven road miles [8]. In addition, physical testing of self-driving cars is difficult, expensive, and potentially unsafe because the autonomous systems of the car must be able to handle many driving situations, including corner or thorn cases [8]. Although simulation provides some answers to the development of autonomous vehicle, the safety and security requirements, the physical testing, and the complexity of the software application all pose challenges to the deployment of self-driving cars.

The effects of complexity, which may be deleterious on engineering systems, are as follows:

- Makes the design process difficult because of the uncertainty in the acquisition environment.
- Increases product life cycle cost because the system requires more time and effort to build.
- Makes the job of providing safety assurance more difficult for a safety inspector because emergent behavior can occur, and system behavior can be unstable and unpredictable.
- Makes a system harder to maintain and repair because the system is harder to understand.
- Makes it difficult to report and diagnose problems.
- Makes it difficult to adhere to a task's rules and requirements due to decentralized control and distributed processes within the system.
- Makes it difficult to break down a problem into sub-problems because the system has so many interdependent pieces.
- Makes project planning and risk management more difficult because the system shows dynamic patterns of behavior, which may be unclear in cause and effect.
- Reduces confidence levels in the results of verification and assurance because the system may be chaotic, unstable, uncontrollable, and unpredictable.
- Makes process integration more difficult because of the presence of a system of systems with many stakeholders.
- Makes coordination much harder because of system scale and variety, and stakeholders may hold diverse political views.

However, complexity not always hinders the world in which we live. It sometimes benefits us. In fact, some manufacturers require certain degree of complexity in their supply chain to maintain productive operations. For example, a manufacturer of high-tech hardware has redundant sources of supply to prevent periodic supply disruptions, but this adds substantial complexity to the supply chain [9].

To gain some insights about the challenges of problem complexity, we need to understand the problems arising from complexity. Some examples of problems that arise from complexity are as follows: (1) difficulty in understanding and modeling the system, (2) unpredictable behaviors and emergence, (3) inefficient processes that could damage a company's performance, (4) increasing regulation, (5) weak risk management, and (6) difficulty in system development and project management.

So how do we define *complexity*? Scientists in various disciplines define complexity in many different ways. In the next section, we present the definition of complexity used by this research and discuss the differences between complicatedness and complexity.

a. Complexity

We define complexity as the extreme difficulty of describing, analyzing, controlling, and managing a system consisting of many internal components with countless interconnections [10], [11]. People perceive complexity as the difficulty to understand a system and to represent the system in a meaningful and predictable way. For example, when the structure or behavior of a system is uncertain, unpredictable, or difficult to fully understand, we perceive that the system is complex. Complexity is not inherently an undesirable property. Systems require a certain level of complexity to obtain a desired performance. However, excess complexity in a system can contribute to undesirable emergent behaviors (e.g., unexpected errors on the user interface or unintended power overload on the circuit).

In a complex situation, systems engineers can use complexity measures for comparing one situation to another. In an engineering project, systems engineers can use

complexity measures for estimating complexity. For example, decision makers in a military weapon program can use complexity measures as evaluation criteria for trade studies.

That said, complexity and complicatedness are not the same things. Complicatedness refers to something that is difficult but not impossible to explain and understand [10]. Complicated systems have many intricately moving parts and linear feedback loops that tend not to interact with each other, but they work in predictable ways. For example, a Rolex watch is complicated because it has many intricately moving parts and these parts have many possible interactions among them. In addition, complicated systems are predictable and manageable. For instance, fixing the transmission of a car is complicated, but the steps to drive a car are easy and routine. We can control complicated systems and analyze them piece-by-piece. Hence, we can fully understand and model complicated systems. So, two discerning differences between complexity and complicatedness are (1) the ability to understand and explain the interactions of the parts, and (2) the ability to reliably predict system behavior.

Complexity may occur in many contexts, both in natural systems (e.g., birds in a flock, fireflies flashing synchronously at night, and schools of fish in a coordinated movement) and engineered systems (e.g., traffic control systems, F-35 fighter jets, and self-driving cars). Complexity occurs throughout a system's life cycle, from writing requirements, design, and development to integration, production, and testing. Complexity is present in organizations, business policies, contract management, mergers, acquisitions, supply chains, and distribution networks. Complexity also is present in the system operational environment, system design, and engineering programs. In fact, systems engineers and project managers have to face complex problems and develop complex systems regularly in their engineering practices.

b. Complex Systems

Although many different definitions of complex systems appear in the scientific literature, in this research, we define complex systems as systems with components, interconnections, and interactions that are difficult to relate, comprehend, predict, design,

and manage [12], [13]. Complex systems may display non-equilibrium dynamics as well as emergent, nonlinear, non-deterministic and/or chaotic behavior [12], [13].

Mission engineering and SoS engineering are two areas in which the Navy must confront complex systems. In mission engineering, the Navy applies SE processes and knowledge to design missions involving multiple platforms (e.g., F-18 and P-8) and assets (e.g., GPS and satellites) to complete a mission in the face of adversary actions. The mission is complex because it contains multiple platforms and multiple assets. A SoS is a system that consists of other dependent systems networked together to achieve higher capabilities than the sum of the capabilities of the built-in systems. A SoS (e.g., an airplane, a ship) is complex because these systems are networked together.

c. Project Complexity

This research studies complexity as a critical factor affecting projects in terms of the difficulty of understanding and controlling project risk. Through reviewing the literature, we find that researchers in different fields provide many different definitions of project complexity. In this research, we define project complexity as the degree of difficulty in understanding, planning, scheduling, and controlling project properties as well as project interfaces, consequently making it very difficult to predict project outcomes [14]–[17]. The possible project properties (project duration, project cost, number of requirements) and project interfaces (e.g., suppliers, top management, stakeholders, personnel, information interfaces, geographic interfaces) are identified and drawn from complexity theory, complex system variables, the review of the relevant literature, and subject-matter experts' experience. The identified project properties and project interfaces are then applied to develop the complexity metrics, which serve as metrics of the overall project complexity.

Project risk deals with the unknown and known events that may affect project outcomes. Project teams often make tradeoffs between the triple constraints (schedule, budget, and performance) of project management. Project teams also deal with three aspects of management. First is the project risk because of decision-making under uncertainty. Second is the difficulty in accomplishing project properties such as cost, schedule, system design and

integration, and permit approval from external agencies. Third is the complexity in planning, scheduling, and controlling of project properties and project interfaces. Broadly speaking, we observe that the greater the number of project elements (properties) and their interactions (interfaces) to achieve project outcomes, the higher the degree of complexity of a given project. In that regard, the literature indicates that project properties and project interfaces are two key metrics to measure project complexity [14]–[17].

2. Characteristics of Complexity

In literature concerning the characteristics of complexity, Page [18] identified five characteristics (numerosity, connectedness, interdependence, diversity, and adaptivity) that contribute to whether the system is complex or not. Systems that consist of entities (i.e., parts, components) that are numerous, connected, interdependent, diverse, and adaptive all contribute to making a system complex [18]. Furthermore, Chaisson's work shows a strong relationship between our perceptions of "complexity" and "the flow of energy" [19]. Without "the flow of energy," the system ceases to be complex because the system is in the equilibrium or static state. He suggests using the power P flowing through a mass M as a measure of the flow of energy [19]. Consequently, complexity will continue to increase while increasing the flow of energy [19]. Chaisson also points out that a non-equilibrium system frequently exhibits extreme events such as cascades of collective failure [19]. Because of Chaisson's work, we can add two additional characteristics (non-equilibrium and nonlinearity) to Page's five characteristics of complexity. Hence, we have seven characteristics known as the seven building blocks that contribute to complexity. They are as follows: (1) numerosity, (2) connectedness, (3) interdependence, (4) diversity, (5) adaptivity, (6) non-equilibrium, and (7) nonlinearity. To gain some understanding of these seven characteristics, let us review them in order.

a. Numerosity

Numerosity is a size measure of complexity [18]. A size measure is the number of elements of a system (e.g., number of parts, number of interactions between parts, number of total collection of properties of a system, and number of observable behaviors) [18]. Size measures do not always lead to complexity. For example, the Defense Advanced

Research Projects Agency's (DARPA) [20] found that the presence of many integrated circuits (ICs) in a system does not necessarily increase the complexity of the system because the elements in an IC are all the same. However, the ICs used in today's airplanes, satellites, rockets, and cars have more parts, and these technologies are getting more complex. To improve performance and reduce cost, an IC has many components packed tightly, and, therefore, the number of unanticipated interactions grows substantially. Hence, the increasing number of ICs in a system is likely to increase complexity. According to the DARPA META program [21], complexity has increased for ICs. Likewise, the complexity has also increased for software-intensive systems [22], [23] because the number of lines of code has increased for these systems. Most size measures appear as estimators for complicatedness. However, complex things will typically have a large number of parts and a large number of possible behaviors. Thus, size measures play a role in the degree of complexity.

In terms of size measures related to the number of possible behaviors, we know that multiple people form complex systems. A large number of people form a society. Consequently, we observe a large number of possible behaviors in a society. For instance, a couple in a family cooperates on house chores. When children and grandparents also live in a household, everyone in the house collaborates on house chores. Therefore, we notice some changes in behavioral complexity (e.g., fewer chores and more interactions for each person). Now, consider multiple households in a village. All of the intra-household behaviors persist, but now there is additional cooperation and competition between the households. The village may also institute some form of government to provide order and stability. Subsequently, we observe some changes in behavioral complexity. Villages grow into a town. Towns expand into a city. Cities become megacities. Multiple megacities form states. Multiple states become nations. Multiple nations become a continent. Continents cover the planet. With each increase in aggregated population, new behaviors arise. In short, numerosity contributes to increases in complexity.

From a project perspective, numerosity appears to loosely associate with the size of development effort of a project (i.e., labor hours). One way to measure development effort is in terms of how long and how many people it takes to complete the project. The

Constructive Systems Engineering Cost Model (COSYSMO) [24] is one way to accurately estimate the amount of effort associated with performing the system engineering tasks in projects. Hence, we might be able to use COSYSMO as a metric for project complexity. We will have in-depth discussion of COSYSMO in Chapter III.

b. Connectedness

A network consists of nodes and links. Nodes are things (e.g., systems, people, computers, and machines). Links are relationships between the nodes (e.g., friendships, communication lines, and business dealings). Whenever we have two or more nodes interconnected by links, we have a network. Figure 1 shows an organizational network where the entities are organizational units, the connections are information flows (i.e., status reports and decision notifications), and the interactions represent the exchange of information.

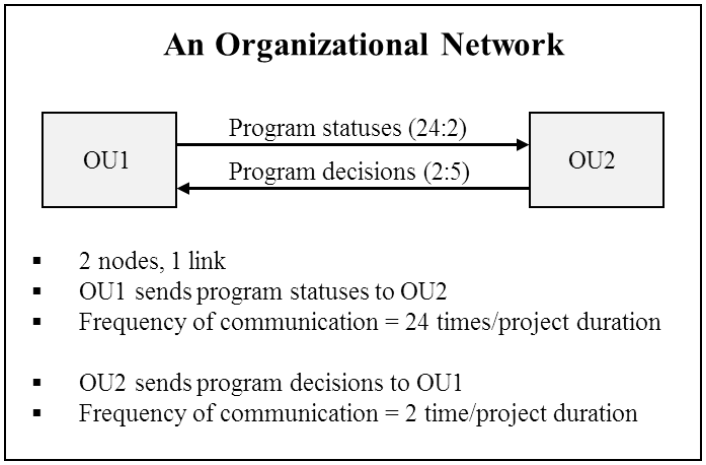

Figure 1. An example of an organizational network

In systems engineering, we measure connectedness by measuring the interactions between systems. Examples of these interactions are the transfer of energy, information, money, and material. One way to measure connectedness in complex projects is to develop a network diagram between interdependent organizational units and show the relationships associated with those organizational units. By analyzing the frequency and importance of interactions between each organizational unit, we can determine the connectedness value

of the project. For instance, as shown in Figure 1, we have a project that involves two organizational units. There are 2 nodes and 2 links that connect in an organizational network. OU1 transfers the information 24 times to OU2, and each transfer has a level of importance of 2. OU2 transfers the information 2 times to OU1, and the transfer has a level of importance of 5. The connectedness of the project can be defined as the sum of each weighted transfers of OU1 to OU2 plus the sum of each weighted transfers of OU2 to OU1. In this case, OU1 has 24 transfers, and each transfer has a weight factor of 0.4 (2/5). OU2 has 2 transfers and a weight factor of 1 (5/5). Thus, the connectedness of the project is 11.6 (24 times 0.4 plus 2 times 1). In short, complex projects will usually have many possible interactions and behaviors, and the number of interactions and observable behaviors is an indirect measure of connectedness.

In many complex projects, risk factors (i.e., lack of management support or mediocre team performance) are identified as inherent complexities of a project [25], [26]. For example, lack of management support can create uncertainty about the project, which can affect team performance and the organizational structure of the project in terms of who has authority over the project and who makes the final decisions. Senior management adheres to a complicated decision-making process that could delay project decisions and increase the risk of schedule delay. They can demand rigorous project oversight that could interfere with team performance. Project managers have to cope with senior management's decisions. Similarly, lack of coordination among organizational units (i.e., contractor unit, project team, and integrated product team) can cause schedule delay. Hence, by definition in this research, project complexity is an indication of project risk.

c. Interdependence

From a systems engineering perspective, interdependence is an informational relationship, a control relationship, or a resource relationship such that one organizational unit depends on another organizational unit [27]. It emerges due to interactions between elements of the engineering project. In engineering design, organizations are in charge of executing the tasks. In this perspective, interdependence is the degree, in terms of behaviors and results, to which an organization depends upon the actions of another organization

[27]. As a result, we have a set of tasks among the organizational units. We need a set of capabilities to execute a task. Furthermore, because an entity in an organization that has those capabilities may apply and complete the task, we can define interdependence as emerging between tasks rather than between organizations [27]. In general, interdependence is high when the coordination load is high, which might require more time and effort to manage these tasks [27]. However, in systems engineering, management can change roles and responsibilities of each organization and limit the capability of task in each organization [27]. As a result of the reassignment of roles and responsibilities of organizations, the interdependence among organizations changes, but the "interdependence between tasks" remains the same [27]. Hence, we have to examine interdependence both in task and in organization.

Giachetti [28] proposes a model and measurement formalism based on the notion of process and organization to analyze interdependence in enterprise systems. He creates a process control diagram that includes the frequency, the level of importance, and timing of the flow of information to represent the strength of three different interdependencies: a sequence of a task, the reciprocal of a task, and collective resources. One drawback of this model is that it omits informal communications between processes and organizational units. Nevertheless, this model is a reasonable way to measure interdependence in enterprise systems.

In sum, systems engineers could model the measure of interdependence possibly using an organizational network diagram, which comprises of organizational units and the frequency and importance of the flow of information between each organizational unit.

d. Diversity

Page [12] defines diversity as the lack of sameness. In other words, the number of different types of elements determines diversity. Diversity can mean variation in parameters and attributes (e.g., beak depth in Darwin's finches), multiplicity of types (e.g., models of cars), differences in populations (e.g., the number of employees in each organization), differences in structure (e.g., software architectures), and differences in function (e.g., a specialty either in air conditioning repairs or automotive transmission

repairs) [29]. When present, diversity appears to increase the number of behaviors, which in turn increases complexity. Numerosity contributes to complexity most when the elements are diverse. In short, diversity is a factor affecting complexity. For example, as shown in Figure 2, a hierarchical measure of complexity takes into account the number of layers and the diversity of differing structures. Given the same number of layers, the structure with more diversity is more complex than the structure with little diversity. Diversity in the elements (e.g., layers, structures, and people) in an interdependent system is likely to foster differences in outcomes, and, therefore, increased complexity. Complex systems tend to have much more diversity [12], [29].

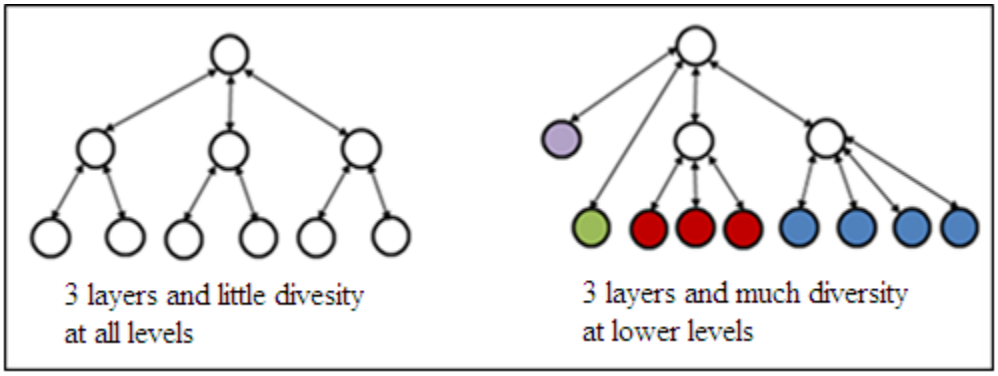

Figure 2. Two different hierarchies with two different diversity values. Adapted from [29].

Three diversity measures appear to be applicable to engineered systems. They are: (1) variation measures, (2) distance measures, and (3) attribute measures. Variation measures describe diversity within a type. Distance measures tell the magnitude between two different types [29]. Attribute measures determine "the total number of attributes" required to describe "all elements of the system" [29].

In complex projects, diversity of team members may involve differences in history (e.g., past experience, education, and job assignments), preference (e.g., likes versus dislikes), personality (e.g., logical versus emotional), and adaptability (e.g., ability to learn) [29]. Based on the definition of diversity of complex projects, diversity appears to be loosely associated with key risk factors of a project. For example, in a diverse team, the

variations in team-related issues (e.g., team conflicts, staff turnover, lack of motivation, and lack of team communications) can be quantified by a variation measure. Moreover, different types of job positions in interdependent organizational units are likely to foster differences in job assignments (more task types) and outcomes (more behavioral types), and, therefore, increase complexity. From this perspective, systems engineers could possibly model the measure of diversity in engineering projects using different types of job positions in organizational units.

e. Adaptivity

In SE, adaptivity is a system's ability to change its function or behavior in response to changes in its environment [29]. When present, adaptivity appears to increase complexity. Adaptivity may be random such as in transformation or variation. For example, search engines have search and indexing algorithms that allow users to search efficiently (i.e., eliminating redundant links or reducing the number of retrieval links, saving users' time from useless page searching). A browser plug-in contains re-search and re-classification algorithms to refine the results returned by the search engines so that users obtain the closest links to their search text. The re-search and re-classification algorithms contain certain kinds of selection and decision-making, which are the two elements of adaptivity. Adaptivity may be specific, as produced by learning. For instance, in an adaptive learning application, the application uses an artificial intelligence (AI) algorithm (e.g., machine learning [30] and deep learning [31] technology) to customize the content, curriculum, method, or pace of learning for each student. The AI algorithm contains certain kinds of profiling, learning, selection, and decision-making, where all are elements of adaptivity. Another example of adaptivity is a dual-mode cellphone that can automatically choose a wireless standard at its current location.

In complex systems, adaptivity is difficult to quantify because it potentially involves every element of a system and every activity the system can perform. However, we know that some adaptive mechanisms are more "complex" than others. We could possibly rate the relative complexity of the system's adaptive mechanism based on a Likert

scale [32]. However, this approach has not been proven through validation in empirical research studies.

In engineering projects, adaptivity requires some sort of change in the project's properties (e.g., project cost, project duration, and the number of requirements) or in the project's interfaces (e.g., suppliers, top management, and personnel). For example, a project manager decides to compress the project schedule to meet his allocated budget. The decision to change the project schedule is an element of adaptivity.

Based on the definition of adaptivity, we can derive five characteristics of adaptivity as follows [29]: (1) A change in the ability of an element to affect or to be affected by other elements, (2) A change in the connections, interactions, or relationships between elements, (3) A fluctuation in the amplitude or frequency of a function, (4) A change in the function of an element, and (5) A change in the design of an element.

In complex projects, we could possibly model adaptivity by using key complexity characteristics and loosely associate them with adaptive mechanisms. This approach is very similar to the approach of measuring adaptivity for complex systems. Nevertheless, this approach has not been proven through validation in empirical research studies.

In software development projects, Subramanian and Chung [33] propose using the architecture adaptability index (AAI) and the software adaptability index (SAI) for measuring software adaptivity. They define software adaptivity as the ability to accommodate changes in the software environment. Some changes in the software environment include the changes in the number of inputs and changes in non-functional requirements (i.e., performance, maintainability, and security) of a software system. Non-functional requirements are qualitative requirements such as a 100% reliability requirement for an aircraft engine or a software requirement to use micro-service principles for self-driving cars. Based on the definition of software adaptivity, Subramanian and Chung [33] suggest that different software architectures have varying degrees of adaptability, which we measure by AAI. They claim that the closer AAI is to one, the more adaptable the software architecture is [33]. Different software programs have different degrees of adaptability [33], which we measure by SAI. Subramanian and Chung claim that the closer SAI is to one, the more adaptable the software program is [33]. However, the definitions

of AAI and SAI are limited to measure adaptivity for non-functional requirements, and software can only satisfy non-functional requirements within acceptable limits.

f. Non-equilibrium

A non-equilibrium system is a system that dissipates energy persistently regardless of the amount of energy supplied from the external systems [34]. For example, the combustion reaction in a gasoline engine of a car is a non-equilibrium system. The work of Chaisson shows that as the flow of energy of a system increases, the more capable the system is of reducing entropy, and thus, the more complex the system is [19]. A non-equilibrium system is a dynamic system and often exhibits extreme events (i.e., cascades of collective failures) [34].

Intuitively, the flow of information is associated with the dynamics of the system, and dynamics are related to either information processing, work performed, or the flow of energy. Clark and Jacques [35] propose that one possible way to determine complexity in dynamic systems is to observe changes in kinetic energy. This measurement provides insights into system risks and why some systems fail in certain ways [35]. This approach is still in its infancy and requires further study. In short, our understanding of non-equilibrium systems is still very limited.

g. Nonlinearity

Nonlinearity refers to a relationship in which change in any of the inputs results in a disproportional change in the outputs [36], [37]. For example, a modest reduction of airfare between cities can disproportionately increase the number of passengers. In mathematics, a nonlinear function is likely to have an exponent, a maximum, and a minimum. A small stimulus input may produce a very large response in a system. Similarly, a rather large stimulus may produce little response. In other words, a stimulus to one part of the system may produce nonlinear responses in the system. We cannot predict the behavior of a nonlinear system from the behaviors of the parts [38]. For example, in a manufacturer of automobile parts, the variance between sales and orders tends to increase as this information sends to upstream inventory and production decisions [39]. This phenomenon is called the “bullwhip” effect [39]. From a systems engineering perspective,

project development elements (e.g., requirement, user involvement, and management support) and processes (e.g., regulation, decision-making, and technology integration) may be nonlinear because their effects may not be proportional to their causes.

3. Characteristics of Complex Systems

In Section 2, we discussed seven characteristics of complexity at the component level of the system. However, at the system level, the system exhibits additional characteristics of complexity. In literature concerning the distinction between systems that are complex and those that are not, the following are five characteristics of complex systems:

- "Self-organizing behavior" [40] (e.g., the Mars Exploration program to understand Mars' environment),
- Evolution and adaptation to environment [41] (e.g., the SpaceX Mars program to land a human on the surface of Mars),
- "Evolving problems" that create "constantly changing needs" [40] (e.g., the California high-speed railroad project [42]),
- "Decision-making under uncertainty" [40] (e.g., the Apollo program), and
- Nonlinear cause and effect such that small disturbances create large effects [41] (e.g., the 2010 Deepwater Horizon oil spill in the Gulf of Mexico [43]).

In parallel to the five characteristics mentioned in above, Sheard [40] presents the following four characteristics of complex systems: (1) self-organization, (2) evolutionary dynamics, (3) uncertainty, and (4) nonlinearity. Moreover, scientific literature on complexity includes some additional characteristics of complex systems as follows [40]: (1) emergence, (2) nonlinearity; (3) limited predictability, (4) evolutionary dynamics, (5) self-organization, (6) uncertainty, and (7) spontaneous order complexity. To gain some understanding of complex systems, the first step is to review the definitions of these seven characteristics.

a. Emergence

Emergence describes “the whole” as “different from the sum of the parts” [41], [44], [45]. For example, the smell of ammonia exhibits properties from the whole compound of nitrogen and hydrogen. Emergent behaviors are usually unexpected when first observed. For example, many discrete pixels produce an emergent image of a motorcycle. There are many paths to emergence. Hence, there are many types of emergence. For example, a rack and pinion gears system that converts rotary motion into linear motion can be described as linear emergence. Linear emergence usually arises from traditional systems engineering. On the other hand, a nonlinear system such as an electronic circuit that converts a square wave into a sine wave can be described as nonlinear emergence. Thus, nonlinear emergence arises in nonlinear systems, and nonlinear systems are complex. Emergence is likely to pose many challenges to systems engineers because emergence in a system is unpredictable, and, therefore, its behaviors are difficult to control. Many nonlinear systems exhibit emergent behaviors. To design and analyze complex systems, engineers may need to do the following [29]:

- Predict emergence where possible.
- Accommodate emergence when it appears unpredictably.
- Control emergent behaviors, and even design systems to produce certain kinds of emergence.

b. Nonlinearity

In complex systems, irregular behavior arises from nonlinearity rather than from random driving forces [46]. For example, in a regional weather system, the movement of the jet stream affects the air pressure and helps shape the weather in that region. It is difficult to determine design parameters when the system is nonlinear. Whenever parts of a system collaborate or rival with other systems, nonlinear interactions occur. There are degrees of nonlinearity. Too much nonlinearity is likely bad for a system because the system may become uncontrollable. To design and analyze nonlinear systems, engineers may need to perform the following [29], [46]:

- Treat any nonlinearity system as a minor perturbation to a linear system.
- Redesign systems that cannot make linear approximation.
- Control or minimize nonlinearity behaviors by designing adaptive systems to accommodate certain kinds of bifurcations and to limit abrupt change in a system.

c. Limited Predictability

Limited predictability refers to the minimal capacity to predict in advance a system's state or behavior [47]. For instance, in a nonlinear system, small changes in initial states can cause very different dynamics over time, and, thus, long-term predictability is not possible [46]. Thus, when contending with complex systems, any predictive activity is an educated guess at best. For example, weather forecast systems can predict the weather within a few days with a limited degree of accuracy. From a traditional SE perspective, SE success mostly depends upon the repeatability and predictability of system behavior. For instance, in an ordinary linear system, systems engineers can analyze the behaviors of parts to predict the behavior of an entire system. On the other hand, complex systems are inherently unpredictable. To overcome or accommodate challenges posted by complex systems, systems engineers likely need to do the following:

- Study and gain in-depth knowledge of the system.
- Conduct modeling and simulation to establish key parameter values.
- Determine or estimate a time horizon at which the system's behavior trajectories diverge.

d. Evolutionary Dynamics

Evolutionary dynamics refers to a system's structure or behavior that constantly changes and innovates over time [41]. Evolutionary dynamics often occurs in complex adaptive systems. For example, although the human immune system has relatively few cell types, it is capable of responding to many threats. Another example would be large

economies in which millions of buyers and sellers trade hundreds of thousands of commodities. Although each agent acts in his or her own best interests, economies normally exhibit stability and usually show steady, long-term growth. In short, the human immune system and the economic systems seem to exhibit evolutionary dynamics. The human immune system and the economic systems improve on its own.

On the other hand, some man-made systems do not have this characteristic. The F-35 fighter jet, for example, is not evolutionary. Although it appears that the design of each new fighter jet becomes better (e.g., F-14, F-18, F-22, to F-35), the evolution of fighter-jet technology is something imposed by the human designers.

However, a biomechanical system such as a prosthetic arm is an artificial intelligence system that exhibits certain characteristics of evolutionary dynamics. For instance, human movement can be represented by nonlinear responses [48]. The stretching of human tissues show nonlinear responses [48]. The prosthetic arm has electrode arrays of sensors that interpret hand movement patterns [49]. The algorithm that analyzes and interprets the hand movement patterns is based on a dynamic systems approach, which is a nonlinear model [49]. As the speed of the computer improves over time and the models for dynamic systems have been enhanced and optimized, the algorithm becomes better and more accurate in interpreting and predicting the hand movement's positions in 3-D. Hence, a prosthetic arm has evolved and become a better system for movement control and extend-to-lift maneuver. The movements of a prosthetic arm look natural and smooth. To strengthen evolutionary dynamics in systems engineering, systems engineers may perform the following:

- Encourage competition and cooperation in research among project teams and organizations.
- Facilitate the process of natural selection (adaptation).
- Emphasize social networking to encourage innovation.

e. Self-Organization

According to Heylighen [50], self-organization appears when elements of a system collectively form certain patterns. It occurs from the bottom up organizing unit. For example, synchronized flashing of fireflies at night exhibits the capacity of self-organization. Crystals form from atoms by self-organization. Other examples of self-organization include the movements of flocks of birds, herds of horses, and schools of fish. Self-organization is also applicable in engineered systems. An electronic hardware based on a programmable chip and "self-evolvable" communicating components is an example of a man-made system that exhibits self-organization [51]. Self-organization often leads a system to critical states with tipping points that could trigger catastrophic events [52]. To minimize systems reaching the tipping points, systems engineers can do the following:

- Reduce random noise.
- Conduct modeling and simulation to predict or identify areas of potential positive feedback.
- Control the inputs by limiting strong fluctuations and overloading.

f. Uncertainty

Uncertainty refers to decision-making involving unknown or incomplete information [53]. One way to measure uncertainty is to use "a set of possible states or outcomes where probabilities are assigned to each possible state or outcome" [53]. For example, when a vending machine spits out a coin, it is uncertain whether it will land on heads or tails. Nevertheless, we know the probability of tails is 0.5, though the actual value is uncertain. Uncertainty is also applicable in engineering. Uncertainty due to either probabilistic system behavior or insufficient knowledge contributes to problem complexity [54]. To minimize uncertain dynamics of a system, systems engineers can do the following:

- Conduct modeling and simulation of SoI to predict or identify areas of variations and nonlinearity.
- Control the inputs by rejecting or limiting disturbances.

g. Spontaneous Order

Spontaneous order is a process that creates a meaningful structure by independent agents acting in self-interest without coordination [50]. It is an emergent phenomenon that occurs both in natural and engineered systems. Spontaneous order has a number of phenomena because a system's behavior is not just different from the component behaviors, but it is also different in kind. For example, crystals form from the order of atoms. The international monetary system is an example of a man-made spontaneous-order system that provides temporary credit to consumers undergoing a current-account deficit [55], [56]. To foster spontaneous order, systems engineers can carry out the following:

- Plan and support organizations.
- Emphasize rules and standards.
- Cooperate with individuals in project teams and organizations.

Learning the characteristics of complexity is just the first step in studying complex systems. The next step is to study the causes and effects of complexity on engineered systems and engineering projects. This is important because complexity is an indicator of project risk, and systems engineers often need to manage this risk. The next section presents the causes of an increase in complexity.

4. Factors that Cause an Increase in Complexity

Describing something that is difficult to predict is a nonlinear problem. Requirements volatility and unforeseen complexity could drive the projects in a non-linear direction. An increase in complexity in a system is due to the following factors [57]:

- Number of parts,
- Number of interconnections,
- Nature of interactions such as nonlinearity or changes that are not constant functions of the input variable, and
- Dynamic shifts or changes as a function of time.

Understanding the causes of complexity in engineering projects is necessary because systems engineers can study these causes and then take actions to mitigate potential project risks such as schedule slippage, budget overruns, and failure to meet requirements.

5. Effects of Complexity on Projects and Systems

The complexities of a project seem to be driven by changes to the scope and unexpected business decisions. Complexity affects a project or a system in the following ways [57]:

- It increases the risk of schedule delay.
- It increases the risk of budget overruns.
- It increases the risk of failure to meet requirements.
- It contributes to higher costs by requiring more effort in systems development, verification, and validation.
- It creates a greater challenge in designing the system.
- It introduces ways in which a system can fail.
- It increases uncertainty of system dynamics.

In addition, when a project is constantly changing the requirements, it is difficult for the project manager to set achievable goals. Evolving cause-effect relationships in a network of systems may lead to concealed bugs and unexpected behaviors [58]. Furthermore, constant change in requirements may create unintended consequences such as system degradations and disruptions and can devastate or even abort the intended actions [59]. Understanding the effects of complexity on projects and systems is important because systems engineers can take actions to reduce or mitigate the complexity of the problem.

B. RESEARCH RELEVANCE AND OBJECTIVE

Project managers, systems engineers, hardware and software developers, and researchers seek to answer the following questions:

- Can we develop a useful measure of complexity for systems engineering?
- Can we use these measures of complexity to determine the level of complexity in an engineering project? From such a measurement, can we plan development efforts and apply resources in appropriate places?
- What impact is a given complexity value likely to have on engineering projects?
- Can we reduce or mitigate complexity? If yes, what are the methods or techniques for reducing complexity?

At this time, there are no widely agreed upon or standardized complexity measurement standards for engineered systems and engineering projects. The absence of complexity measurement standards opens up the opportunity for this research. In short, useful measures of complexity in engineered systems and engineering projects appear to be the most-requested standards by systems engineers [40].

This research has the following three objectives:

- To provide practical and effective measures of complexity of engineered systems and engineering projects for systems engineers by introducing a model for complexity analysis,
- To validate this model of complexity measurement based on empirical data from three Navy software projects, and
- To use the results of complexity measurement to determine ways for reducing complexity.

Ultimately, systems engineers can use the model to determine system development approaches and organizational strategies to reduce project complexity.

C. CONTRIBUTIONS AND BENEFITS TO THE BODY OF KNOWLEDGE IN SYSTEMS ENGINEERING

Numerous studies on software complexity focus on the size of the code, control flow, data flow, the size of the project team, the number of interdependent programming tasks, and other performance parameters. This research extends the complexity measurement in software engineering by bringing together into one holistic model that consists of several different metrics proposed by several different researchers. This model includes measures of number of personnel required for the development effort, defect density, number of geographical distribution of project teams, organizational complexity, requirements volatility, and the number of different types of job positions. This approach incorporates principles of risk management and project management in software engineering by relating them to risk analysis and decision-making to support high quality systems engineering.

In addition, this research covers a use case of SE body of knowledge in the topic of SE advanced measurement where systems engineers and project managers use the PCMM and associated methods to measure complexity and assess project risks to ensure affordability. For example, the PCMM link project complexity to systems development cost, schedule, performance, and risk. As a result, the PCMM provides project managers and IPT management teams with common practices in the areas related to the review of work products, roles and responsibility expectations, accountability demands, and affordability.

The main contributions of this research are as follows:

- A model for identifying key complexity measures in engineered systems and relationships between different elements of complexity within systems,
- Potential approaches for complexity reduction,
- A systematic method for measuring system and project complexity and a demonstration of the model using empirical data from three Navy software

projects to determine the ranges of values in the proposed complexity measures, and

- A demonstration of complexity reduction using results from the complexity measurements of three Navy software projects.

D. OVERVIEW OF THE WORK

This dissertation is organized into five chapters. In Chapter I, we identify the objectives, potential benefits, and anticipated contributions of the research. The chapter focuses on the following:

- Introduces the notion of complexity as applied in engineered system.
- Describes a scenario of a complex system, problems that arise from complexity, and the importance of complexity measures.
- Identifies several characteristics of complex engineered systems.
- Defines complexity and complex systems.
- Introduces the seven building blocks of complexity that form the foundation of this dissertation.
- Includes several research questions and reasons for a need to answer these questions.

Chapter II presents measures of complexity drawn from previous literature. This information helps form a foundation to develop a model of useful measures of complexity. Chapter II consists of the following:

- A broad overview of complexity science history and technologies,
- A discussion of recent developments in measures of complexity of engineered systems,

- A description of several measurement methods for complex engineered systems as well as an evaluation of their advantages, disadvantages, and relevance,
- A set of ideas, definitions, and properties of complex systems as well as examples of measures of complex engineered systems, and
- An explanation of tools used in complexity research and an assessment of their advantages and disadvantages.

In Chapter III, we present a model of useful measures of complexity. This chapter presents the formulation of the system complexity metrics, system and project properties, and analysis methods used throughout the dissertation. The chapter accomplishes the following:

- Identifies factors that describe project complexity based on key risk factors in a project.
- Creates a new approach to develop complexity indicators, which serve as qualitative measures of the associated factors that describe project complexity.
- Develops the model and methods that estimate complexity of systems and projects.
- Proposes methods for reducing or mitigating complexity.

In Chapter IV, we use empirical data from three Navy software projects (named in this dissertation as projects B4, B6, and B8) to validate the proposed complexity measurement model. The empirical evidence is drawn from project-level observables such as requirements volatility, defect density, development cost, and frequencies of the flow of information between organizations. Chapter IV achieves the following:

- Illustrates validity and relevancy of the proposed complexity model through results from three software engineering projects.

- Addresses concerns and issues of complexity measurement for each engineering project.
- Presents lessons learned from three software projects and proposes recommendations for reducing complexity.

Finally, in Chapter V, we conclude this dissertation by presenting a summary of this research, a list of this dissertation's contributions to the bodies of knowledge, and ideas for future research that were inspired by this work. In addition, we discuss the effects of complexity on system development efforts.

THIS PAGE INTENTIONALLY LEFT BLANK

II. LITERATURE REVIEW

It's very hard to do science on complex systems.

—John Horgan [60],
"From Complexity to Perplexity"

As noted in Chapter I, the system-of-interest (SoI) for this dissertation is the DOD software acquisition program for the design and development of complex systems typically used in defense acquisition. No single source, but rather a combination of the several sources of complexity, appears to increase complexity. This realization solidifies the need for creating a complexity profile that consists of multiple complexity measures for project risk assessment. Thus, this chapter starts with a discussion of sources of complexity and system complexity as a risk in project management, systems design, manufacturing, testing, and deployment. Next, we present a background and history of complexity science. The subsequent section reviews recent developments in measures of complexity of engineered systems, including metrics to measure complexity in systems design and development, definitions of complex systems, and properties of complex systems as well as examples of measures of complexity. Drawn from previous studies, this section introduces the seven elements of complexity that will form the foundation of this dissertation. Finally, we identify certain gaps in the literature concerning complexity measurement and propose several solutions to address these gaps.

A. SOURCES OF COMPLEXITY

Today's engineered systems such as offshore wind turbines, fighter aircraft, satellite communication systems, and control systems for smart power grids are complicated, complex, and challenging to design and manage. They are large systems, difficult to maintain and prone to unanticipated failure, and they carry huge costs and risks. Keeping complexity under control in these systems is necessary to improve their affordability. In a 2008 RAND Corporation report [3], roughly two thirds of the overall cost escalation for fighter aircraft was attributed to system complexity (Figure 3). Some government-funded fighter-jet programs (e.g., F-22A and F-35) have been delayed or

cancelled because of cost escalation. Many project managers often need to make tradeoffs between functionality and performance of a system to limit a system's complexity and to improve system affordability. To constrain and manage complexity, systems engineers need to identify useful measures of complexity.

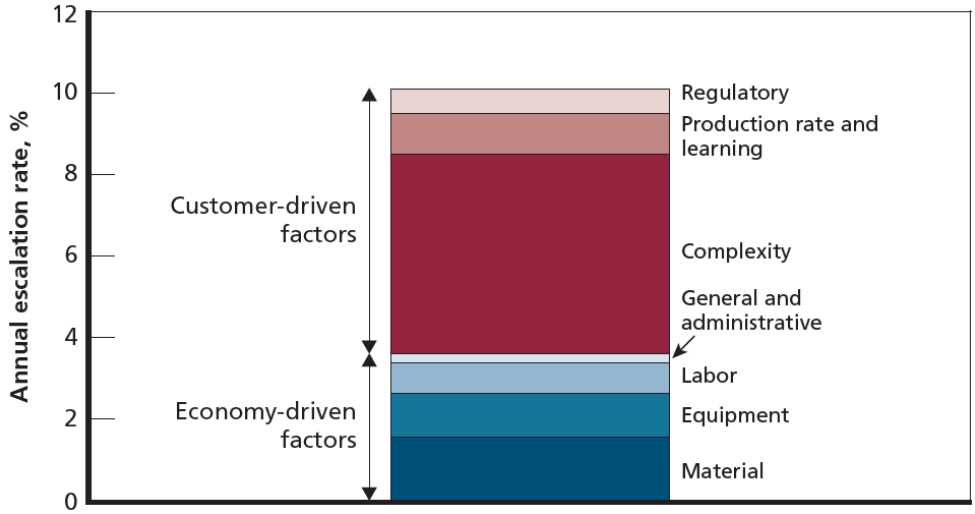

Figure 3. System complexity contributes about two thirds of the overall cost escalation for fighter aircraft. Source: [3].

In general, increased complexity contributes to higher costs, greater challenges in systems design, and more effort required in systems development, verification, and validation. Measuring complexity is an important issue in project management and in the design, manufacturing, testing, and deployment of systems. In addition, complex systems may have emergent behaviors (e.g., unexpected errors on the user interface or unintended power overloads on the circuit) during system validation and operation. These potential behaviors can lead to undesirable and expensive system recalls and redesign.

Systems engineers must identify the nature of complexity and the extent of complexity that imposes on the development program [57]. Such an analysis will help systems engineers to determine whether any mechanism would exist for reducing complexity such as by decoupling the system from its environment. According to the International Council on Systems Engineering (INCOSE), complexity in a development

program can arise from at least three sources [58]: (1) the problem being addressed, (2) the system environment, and (3) the system being developed. In addition, engineering processes themselves are also sources of complexity [61]–[64]. Therefore, to quantify the complexity of engineered systems, systems engineers need to understand complexity of the problem, environment, system, design of the system, and the engineering process. Let us discuss each source of complexity in order.

1. Complexity of the Problem

In engineering projects, systems engineers commonly observe several sources of complexity of the problem. Understanding these sources of complexity of the problem will be valuable to project managers for project planning. According to Sheard and Mostashari [65], complexity of the problem can be the result of the following underlying issues:

- Ill-defined problems,
- Conflicting requirements,
- Too many problem elements,
- Interactions between problem elements,
- Diverse and demanding stakeholders,
- Interacting stakeholders,
- Resources issues,
- Multiple constraints, and
- System of systems orientation.

However, people are often the root cause of some of these issues due to negligence or lack of emphasis.

2. Complexity of the Environment

Many projects often encounter system environment problems. From our SE experiences, we know that system environments are inherently complex [58]. By knowing the sources of complexity of the system environment, project managers can estimate the expected effort needed to complete the project. Some of the complexities of the environment include rapidly evolving threats, strong dependence upon highly variable

environmental conditions, and large disturbances of the environment resulting from resource consumption, waste products, and physical damage. Appreciation and understanding of system environment problems can help systems engineers develop usable measures of complexity.

3. Complexity of the System

Stevens [66] suggests one way to understand complex engineering problems is by studying the four contexts of the system: (1) the strategic context, (2) the implementation context, (3) the stakeholder context, and (4) the system context. First, systems engineers use the strategic context to look at the scope and stability of the intended mission. They need to determine whether requirements are expected to change with time and whether the mission is intended to address a single function or multiple functions within an enterprise. Second, project managers use the implementation context to look at the scale of the effort. They need to decide whether the project aims at developing a single system for a specific purpose or multiple interrelated systems. Third, the stakeholder context addresses potential difficulties with stakeholders. Project managers need to evaluate whether the stakeholders are on board with the stated objectives and whether the relationships with stakeholders are stable or changing. Fourth, the system context addresses the expectations of the system. Systems engineers need to analyze and determine whether the expected system behavior is known or likely to evolve as the design is developed. Engineers study these four contexts to gain understanding of the problem. By determining the range of the system within each context, both systems engineers and project managers can generate a profile of their projects and accurately measure the degree of project complexity.

4. Complexity of the Design of the System

System complexity may also arise from system design. Complexity of system design is caused by excessive functionality, complicated feedback loops, a certain problem that human cause, nonlinear and adaptive elements, and high connectivity and interdependence between elements [29]. Furthermore, system requirements, changes of technology, team dynamics, organizational policies and procedures as well as external

stakeholders are also sources that contribute to complexity. Furthermore, to solve these design problems, systems engineers can do the following [29], [61]–[64]:

- Reduce and remove unnecessary functionality.
- Limit the degree of adaptivity.
- Minimize interactions between nonlinear elements.
- Increase connectivity but reduce and limit interdependence between elements.
- Emphasize system-level optimization.
- Resolve or eliminate conflicting requirements.
- Decouple feedback loops to make a system easy to understand and control.
- Reduce the "human factors" in system complexity.
- Ensure adequate redundancy.

5. Complexity of the Engineering Process

In the engineering process, systems engineers often face the challenge of simplifying the design processes without affecting product performance [28], [199]. They sometimes need to make tradeoffs between efficiency and maintaining control of the design processes. The number of tasks and the frequencies of the communications between tasks often determine the level of complexity of the engineering processes [28], [199]. One aspect of the complexity of the engineering processes is the amount of the communications required between the tasks that require decisions and the analytical tasks [28]. Systems engineers must understand the communication patterns between tasks that need decisions and the analytical tasks in design processes so they can minimize the complexity of design processes. For example, systems engineers can reduce the complexity of design processes by either re-sequence non-coupling tasks or identify and remove weak coupling between tasks [67]. Systems engineers can make design decisions based on the values of design parameters using the users' operational range of values (e.g., a rotor speed of 16.8 rpm of a wind turbine). In sum, the literature has established that systems engineers will find value in tracing complexity to its source so they can address it in systems design and project planning.

B. BACKGROUND AND HISTORY OF COMPLEXITY SCIENCE

Finding the common concepts and mechanisms of complexity greatly aids the understanding of complex systems. This section begins with a brief history of complexity science and follows with an overview of the characteristics of complexity and complex systems. We will discuss concepts and methods of complexity science, which will be useful to derive some measures of complexity for this research.

In the 1950s, scientists developed computers, numerical techniques, and algorithms to study nonlinear systems. To analyze the dynamics of nonlinear systems, scientists developed chaos theory. Chaos contains irregular non-repetitive behavior. It represents unpredictability because the cause and effect relationships are unperceivable. The root of chaos is extreme sensitivity to initial conditions. In other words, changes occurring in an initial stage will be amplified and eventually dominate the system. For example, in 1963, Edward Norton Lorenz, a pioneer of chaos theory, discovered the butterfly effect when observing his weather forecasting model [68]. Scientists also observed chaos in semiconductors and electronic circuits. The implication was that nonlinear systems were inherently unpredictable. Thus, chaos theory became an important part of understanding of nonlinear systems. Subsequently, scientists and systems engineers have applied the butterfly wing patterns to models to study nonlinear and complex systems.

In the 1990s and early 2000s, scientists and engineers used network theory to describe complex adaptive systems. A complex engineered system can be modeled as a network; thus, a brief introduction of networks is an essential part of an overview of complex systems.

A network consists of nodes (e.g., people, machines, and computers) and links (e.g., human relationships and communication lines) [69], [70]. A network's function is to transfer goods and services (information, people, raw materials, manufactured products, etc.) between the nodes [69], [70]. Networks can be in flux such that nodes are added and/or deleted [69], [70]. Systems engineers create network models to measure the values of connectedness and interdependence of networks, which are measures of network complexity.

Scientists and engineers developed a few network models to address emergent and adaptive properties of complex adaptive systems. For example, a hub and spokes network has one node that connects directly to every other node along links that comprise the spokes [69], [70]. Airlines typically adopt the hub and spokes model to route all of their traffic through a central or multiple hubs.

Other types of networks include a random network, a clustered network, a small-world network, and a power law network. A random network provides connectivity by creating random connections between nodes [71]. Neuroscientists use the model of random network to study neural networks. In a clustered network, every node is directly connected to many other nodes [72]. Scientists and biologists use this model to study disease spread such as severe acute respiratory syndrome (SARS) epidemics and the HIV/AIDS pandemic [73]. Small-world networks are mixtures of random and clustered networks [74], [75]. Power law networks (e.g., the World Wide Web [76], [77]) have many connections to guard against or prevent random failures. In sum, systems engineers use various types of network models to study complex engineered systems.

Scientists and engineers have observed that diversity has several relationships with complexity. As introduced in Chapter I, Scott Page, an American social scientist, defines diversity as the state or quality of not being the same [15]. In other words, the number of different types of elements determines diversity. When present, diversity can increase complexity [15]. The more different types of elements are present in a system, the more ways they can combine.

Moreover, diversity generally improves robustness and performance [15]. Robustness is the ability to maintain a system's functionality despite a disturbance within it [29]. The more types of elements present in the system, the more functions can be exploited to respond to a disturbance. Robustness requires that there is a restoring response for every disturbance of the system [29].

Many factors can contribute to complexity, including time, space, and interactions. In 1948, Warren Weaver, one of the pioneers of machine translation, distinguished two cases of complexity [78]: organized and disorganized complexity. Organized complexity results from a vast number of parameters describing the aggregate of the system elements [78]. An example

of organized complexity is the design specification of a radar system for air traffic control which has thousands of components. Disorganized complexity results from the sum of a large number of approximately identical system elements, each of which has different values of a few different parameters [78]. An example of disorganized complexity is the behavior of a volume of gas. The gas volume's behavior is describable by a few statistically averaged parameters. However, many man-made systems have system properties somewhere in between organized complexity and disorganized complexity [18], [44].

As stated in the previous section, engineering processes are one of the sources of complexity. In 2006, Minai et al. [79] addressed the challenges that arise in developing complex engineered systems and identified new CSE approaches that are being adopted. Minai et al. [79] suggested that CSE should focus on creating an environment in which systems can develop and expand, change can occur, and selection favors some systems over others. Furthermore, Minai et al. have stated [79]: "Every engineered system is a tool made to serve the ends of its user." This is a basic requirement of engineering. Engineered systems are intentionally designed and fabricated, and they are usually useful. Complex systems include people not only as designers, but also as part of the system (e.g., operations and maintenance). Minai et al.'s work sheds light on the current state of CSE. They recognize the need to enhance the engineering methods of complex systems.

Many scientists are actively pursuing the study of complex phenomena through their observations and hypotheses. Because of significant improvements in computational technologies, scientists and engineers create computational models known as agent-based models in simulated worlds that encompass both biology and technology [46], [80]. In 2007, Miller and Page [46] studied complex adaptive social systems using computational models to explore general mechanisms underlying some complex phenomena in nature.

In 2009, Mitchell [81] presented several thoughtful ideas. She has stated [81]: "Neither a single science of complexity nor a single complexity theory exists yet." This raises an interesting issue in complexity. A theory of complexity needs mathematics that describes complex phenomena and it needs observations and testable hypotheses to explain those observations. A viable theory of complexity (which does not exist yet) would describe and explain such complex phenomena as chaos, order, nonlinearity, criticality,

self-organization, emergence, and aperiodic (but nonrandom) behavior. Mitchell defined a complex system as a large network of components that exhibit nontrivial emergent and complex collective behaviors [81]. Her work refers to nonlinearity, connectedness, interdependence, and adaptivity [81]. These characteristics are part of the building blocks of complexity.

Learning some key concepts and methods of complexity science is useful to derive some measures of complexity for this research. As showed in Table 1, we provide a summary of key concepts of complexity science.

Table 1. Key concepts of complexity science

Concept	Example
Adaptability	Engineers have developed adaptive equalizers or echo cancelers in high-speed data modems.
Self-organization	Software developers have created a mobile game system that can automatically network with nearby game systems to perform a multi-player experience.
Emergence	A face-recognition system processes many discrete pixels and produces a recognizable image.
Attractors (degree of predictability)	Scientists and systems engineers apply the butterfly wing patterns in models of pattern formation for studying nonlinear and complex systems.
Chaos	Scientists have observed chaos in semiconductors and electronic circuits.
Nonlinearity	In AM radio, listeners sometimes experience electronic distortion such as clipping and crossover distortion.
Networks	Airlines route their air traffic through one central or multiple hubs.
Power laws	The World Wide Web is resilient to random failures.

As time progresses, more and more engineered systems will be complex systems, and new tools and techniques for their design, development, and use will need to be developed. Complexity science is moving forward and progress has been made. Available tools and techniques for analyzing complex systems appear in Table 2.

Table 2. Tools and techniques for analyzing complex systems

Tool Set	Purpose
Agent-based Models	• Provide insights about stability characteristics, including rules and constraints of the system. • Show conditions for unstable or dangerous traits of the system.
Network Analyses	• Identify normal (stable) and abnormal (unstable) networks. • Identify network patterns to make predictions of behaviors in networks.
Data Mining	• Finds outliers in data sets that do not fit expected patterns.
Scenario Modeling	• Reflects key dynamics and conditions of systems in some simulated scenarios. • Provides insights into how the systems respond by varying conditions the systems face. Results are helpful for making strategic decisions.
Sensitivity Analysis	• Calculates the ranges of parameters which the system does and does not work.
Dynamical Systems Modeling	• Makes predictions of future behaviors in a system. • Simulates the results of alternative system interventions.

In sum, complexity science is an interdisciplinary field of research that tries to interpret biological and physiological worlds that exhibit capacities of self-organization and adaptation via learning or evolution [81]. This research attempts to refine methods of complexity science.

The world has always been viewed by observers as complex, but questions linger as to what to do with complexity, and how to interpret and measure it. The next section presents previous work concerning several quantitative and qualitative measures of complexity.

C. PREVIOUS WORK

Many scientists and engineers have attempted to develop measures of complexity because complexity has caused project failures, cost overruns, and schedule delays [3]. This section identifies recent developments in measures of complexity of engineered systems and engineering projects.

Through a review of the literature, we find four main types of complexity in engineering projects [14], [81]–[83] represented as follows: (1) structural complexity, (2) organizational complexity, (3) temporal complexity, and (4) technological complexity.

Structural complexity is measured by the number of elements involved in the project and the dependence of these elements relative to each other [82], [84], [85]. It is used for risk assessment in terms of product quality control [84], [85]. It is characterized by the project properties and project interfaces (described either as linear or nonlinear relationships) between each of the organizational units [82], [85]. Project properties contain size of scope, cost of the project, project duration, project team size, and the number of requirements. Project interfaces include defect density and personnel required for the engineering project.

Organizational complexity indicates connectedness and interdependence among organizational units during the system engineering processes [14], [84], [85]. It denotes a measure of the patterns of communication among organizations in terms of their frequency and importance and a measure of geographical distribution of teams [84], [85].

Temporal complexity refers to the uncertainties caused by the changes in the project properties and project interfaces over the course of the project [85], [86]. It is interpreted as a measure of requirements volatility [84]–[86].

Technological complexity relates to the integration of technology in the transformation processes that convert inputs to outputs [14], [86]. It refers to both the variety of some aspect of a task and the interdependency between tasks and between teams [14], [85]–[87]. Elements that contribute to technological complexity include the number of operators, the type of technical expertise, and the type of skilled staff needed [14], [86]. Hence, technological complexity is defined as a measure of the number of different types of job positions [14], [85], [86].

Using the definition of complexity from Chapter I, we have three aspects of complexity described as follows: (1) the degree of difficulty in describing, understanding, verifying, managing, designing, and changing interdependencies of a system, (2) the degree of difficulty in accurately predicting emergent behavior of a system, and (3) the degree of difficulty in explaining emergent behavior of a system. We can apply these aspects to the four main types of complexity to measure complexity in engineering projects.

In general, structural complexity, organizational complexity, temporal complexity, and technological complexity are correlated with one another [85]–[88]. In other words, as the structural complexity increases, the organizational complexity also increases [88], [89]. Let us discuss each of the four types of complexity.

1. Structural Complexity

In general, a system is structurally complex if the interactions between components within the system are difficult to describe or understand. We know that system architecture, software architecture, and software size are project elements that can contribute to structural complexity [17]. Structural complexity metrics are based on the concepts of entropy and logical depth [90]. Through a review of the literature, we find that several researchers have different ways of measuring structural complexity.

In [90], Lloyd's view of structural complexity is the difficulty in describing things. He describes structural complexity as the amount of information required in an event to describe a system [90]. Lloyd's measurement approach of structural complexity uses the Shannon entropy [91] method which computes the average amount of information of an event necessary to describe the system. Shannon's entropy method measures how much

information is contained in the output of a process and is typically measured in bits. Information refers to anything that can answer a question (e.g., digits, text strings, number and type of pathogens in the body, or available food supply around an ant colony). The Shannon entropy measurement method is useful in evaluating the design complexity of any digital signal processing system. In general, for actual systems with many different types of parts, entropy is difficult to calculate because the dynamic range of outcomes (a degree of disorder) may be enormous, and outcomes may be unpredictable. Nevertheless, Lloyd and Gell-Mann applied the Shannon entropy method for measuring structural complexity within the manufacturing process [92]. Similarly, Arteta and Giachetti used Shannon's entropy method for measuring the complexity of business processes [93].

In [94], Kolmogorov proposes an algorithmic complexity measurement based on the size of the smallest computer program that could form a full description of a picture without degradation from noise. In other words, the complexity of how information flows in the system reflects the system's structural complexity. This process is measured in bits. For example, a program that generates a string "MYBIGSTAR" has more algorithmic information-content than the one that generates the string "AAAAAAAA." This measurement approach might apply in medical imaging, acoustic processing, and bio-informatics. However, for large complex systems like a Boeing 787 jetliner, it is not practical to determine its algorithmic information-content because of the enormous amount of information and computation needed to generate the Boeing 787 data model.

In [29], Harney proposes that both an "indentured parts count," commonly called a "bill of materials" [95], and a function count for a system can be ways to measure structural complexity. The bill of materials can be used to measure complexity as a function of the numbers of item types, quantities, indenture levels, and unique part numbers. The list may be useful in determining the structural complexity of past systems, allowing comparisons between programs and the outcomes of various SE tools. A system function count is directly analogous to the method of function point count in software [96]–[98]. First, software engineers analyze and determine the system's functions. After counting each type of function, they can determine an overall system function count by a weighted sum. Since functions are determined relatively early in a design, this might provide early guidance

concerning both the degree of complexity and the actions necessary to accommodate that complexity.

However, there are some drawbacks to these two approaches. For example, the biggest difficulty in measuring a bill of materials is that, because parts lists are not available early in the development process, they are unlikely to be a guide to predicting and solving problems. Furthermore, part count only measures numerosity, and it does not consider diversity, interdependence and other factors that can affect complexity. Similarly, a system function count requires sufficient system definitions to develop system functions. It requires considerable study of prior program efforts and mature requirements to guide that determination.

In [99], Sinha proposes a quantitative structural complexity metric using the following three primary sources of complexity: (1) the complexities of each component, (2) the complexities of each pairwise interaction, and (3) the effect of architecture or the arrangement of the interfaces. To capture the interaction among system components, Sinha proposes the topology complexity metric based on the functional form of the electron energy of an organic molecular system. This metric helps to characterize different types of architectures, and is tractable throughout the system development phase. One advantage of this bottom-up approach is that it can provide complexity estimation of novel components and interfaces where data may not available. One drawback of this approach is that it relies on expert opinion for estimation of component and interface complexities. Another drawback is the lack of a reasonable size of data from real world engineered systems for empirical validation of theoretical predictions. However, this structural complexity estimation method can apply to multiple engineered systems ranging from simple power drills to car engines.

In regard to project complexity, Williams [83] defines structural complexity as the degree of effects on the tasks that is based on the number of elements involved in the project (i.e., the number of hierarchical levels of management, number of organizational units, division of tasks, etc.) and the interdependence (i.e., management interactions and technical interfaces) of these elements relative to each other. He also claims that uncertainty in goals and methods contributes to project complexity due to difficulty in specifying user

requirements and the possibility of change of user requirements after the initial prototypes. We will further review this topic in the project complexity section of this chapter.

According to Simon [100], complex systems are often structured hierarchically with more strong interactions occurring within, rather than between, subsystems. In other words, Simon believes that each subsystem is independent of other subsystems. This idea works like building blocks in that evolution produces complex entities by putting together fewer complex entities. Simon's observations do not reflect the degree of complexity of functions within each component, nor the diversity of differing structures produced by the successive hierarchical layers.

Gell-Mann et al. [92] suggest an effective complexity measure that involves a process of finding regularity and randomness in objects. This approach is to quantify complexity in terms of structure and information within a system. This measurement method computes the amount of information required to depict the regularities of a system [92]. Effective complexity is low for both extremely ordered and very random things [92]. Effective complexity might be useful when comparing two engineered systems to determine if one system is inherently more complex than the other. In practice, though, effective complexity is difficult to measure, and various observers may disagree on the definition of a system's regularity.

A summary of the work of these scholars appears in Table 3.

Table 3. Structural complexity measurements in engineered systems and engineering projects.

Year	Scholars	Structural Complexity Metric
1962	Simon [100]	Measure a system's complexity by the level of hierarchy.
1965	Kolmogorov [94]	Measure algorithmic complexity by bits that represent the content of information of a system.
1996	Gell-Mann et al. [92]	Measure complexity by the structure of a system.

Year	Scholars	Structural Complexity Metric
1999	Williams [83]	Measure project complexity by "the number of elements and the interdependence between these elements" in the project.
2001	Lloyd [90]	Measure complexity by bits that represent the order and disorder of information of the system.
2014	Sinha [99]	Measure the level of complexity based on three sources of complexity: (1) the complexities of each component, (2) the complexities of each pairwise interaction, and (3) the arrangement of the interfaces.
2017	Harney [29]	Measure systems complexities based on a bill of materials and a system function count.

2. Organizational Complexity

Organizational structure often mirrors the system's architecture (called Conway's law [87]). For example, in a system's architecture, we may have systems, subsystems, units, assemblies, subassemblies, and components. This system's architecture mirrors the hierarchical structure of a typical organizational structure (i.e., divisions, departments, branches, sections, teams, and individuals). In general, more hierarchy in the organization will lead to structure that is more hierarchical in a system. In short, organizational complexity is simply structural complexity applied to organizations as the system-of-interest (SOI).

Lloyd defines organizational complexity as the degree of difficulty of describing organizational structure (e.g., a corporation, a division of a company, etc.) and the amount of information sharing within an organization [90]. Measurements include schema length, hierarchical complexity, functional network complexity, effective complexity, stochastic complexity, channel capacity, correlation, and stored information [90]. For example, a

hierarchical complexity measurement indicates the degree of complexity in the vertical and horizontal structure of an organization (e.g., an organization may have divisions, departments, and branches, and each branch may have several locations and buildings).

Baccarini [14] describes organizational complexity as the engagement of a temporary multi-organizational structure to manage a project. He refers to organizational complexity as the level of impacts on the project that is based on the number of project elements (e.g., stakeholders, engineering disciplines, number of employees, business processes, organizational units, and project size) and the relationships (e.g., the number of management and technical interfaces) between these elements [14]. Furthermore, several authors including Giachetti [28], McCann and Ferry [101] as well as Victor and Blackburn [102] have proposed that the interdependent relationships between organizations and business processes can contribute to organizational complexity. Lee and Xia [85] claim that organizational complexity is a result of changes in the hierarchy of organizations, information systems used by the organizations, and business processes. Lee and Xia propose a framework that includes an organizational complexity measurement for assessing information system development projects. Lee and Xia believe that a project environmental conditions (market, social, political, and regulatory conditions) contribute to project complexity [85].

Table 4 shows a summary of the work of these authors.

Table 4. Organizational complexity measurements in engineered systems and engineering projects.

Year	Authors	Organizational Complexity Metric
1979	McCann and Ferry [101]	Measure inter-organizational relations based on a conceptual model.
1987	Victor and Blackburn [102]	Measure the levels of interdependence between work units based on a framework of "alternative conceptualization."
1996	Baccarini [14]	Measure project complexity based on project size and the interdependence of organizational units.

Year	Authors	Organizational Complexity Metric
2001	Lloyd [90]	Measure organizational complexity in terms of the degree of organization based on the vertical and horizontal structure of an organization and the amount of information sharing within an organization.
2002	Lee and Xia [85]	Measure project complexity based on "changes in user information needs, business processes, and organizational structures."
2006	Giachetti [28]	Measure interdependence in enterprise systems based on patterns of workflow and frequency, importance, and delay of the flow of information.

3. Temporal Complexity

A system has temporal complexity if its behaviors are difficult to describe and predict effectively [85], [86]. Furthermore, temporal complexity reflects the level of impacts on the project due to requirements volatility that is caused by the external environment (i.e., environmental regulations, market conditions, political and institutional complexities, funding mechanisms, and technical issues) [84].

Lloyd and Pagels [103] propose a complexity measurement based on thermodynamic depth. This approach suggests that engineers quantify complexity in terms of randomness within a system. This measurement method calculates how many bits of information of the resources are required to describe the particular trajectories leading to the current state [103]. In other words, systems that are more complex are harder to build [103]. For example, when studying a particular organism, a scientist starts with the simplest possible organism, and then determines all the genetic events (mutations, gene doublings, gene modifications, rearrangements, etc.) that lead to the current state of an organism under study [103]. The more events required, the more complex the organism [103]. In practice, this measurement method of randomness within a system is very difficult to apply to natural objects because it is not clear how natural objects evolve to the current state.

Nevertheless, this measurement method appears similar to the Shannon entropy method proposed by Gell-Mann and Lloyd for measuring structural complexity [92].

Wolfram [11] categorizes the following four basic kinds of behavior in a system: (1) equilibrium (i.e., static or simple oscillation), (2) complexity (i.e., aperiodic oscillation), (3) chaos (i.e., unpredictable and aperiodic oscillation with no apparent order), and (4) randomness (i.e., noisy oscillation because of random inputs). Pictures of these four types of behavior appear in Figure 4. From Wolfram's perspective, complexity comprises everything that is not static, cyclic, or random [11]. Wolfram's characterization of these four types of dynamic behavior correlates to the temporal complexity of a system. Based on the understanding of these behaviors in systems, systems engineers have learned to describe temporal complexity in terms of the volatility and predictability of the behaviors of a system.

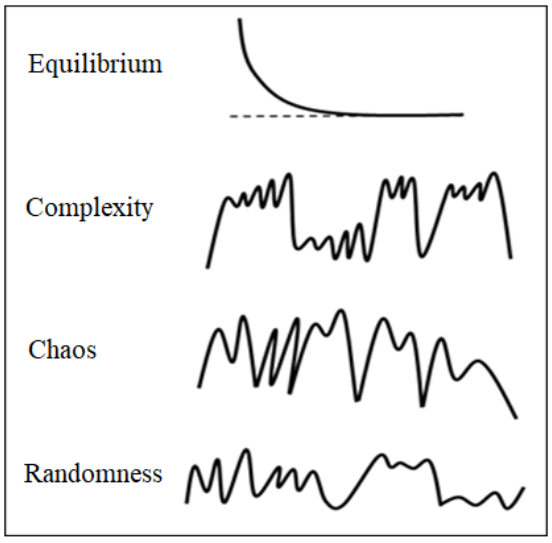

Figure 4. Examples of Wolfram's four types of dynamic behavior. Adapted from [11].

In [104], Perrow proposes a model to characterize complex systems such as air traffic, chemical plants, dams, and nuclear power plants in terms of their risk. He states that complex systems feature feedback loops and tightly coupled interactions between system components [104]. As depicted in Figure 5, nuclear power plants have complex

interactions (i.e., interacting controls and feedback loops) that could result in unexpected behavior. The system also has tight couplings among its components because delays in controls are not possible and the order of operations is complex. Perrow's depiction of tightly coupled interactions between system components reflects that temporal complexity is associated with both the interactions and the coupling of the system components. In addition, Perrow observes that complex systems have catastrophic potential [104] such as a nuclear reactor burning out of control causing a nuclear meltdown. Perrow's model for characterizing complex systems can direct an engineer's focus to dynamic systems and resource allocation in designing these systems—an important practical consideration.

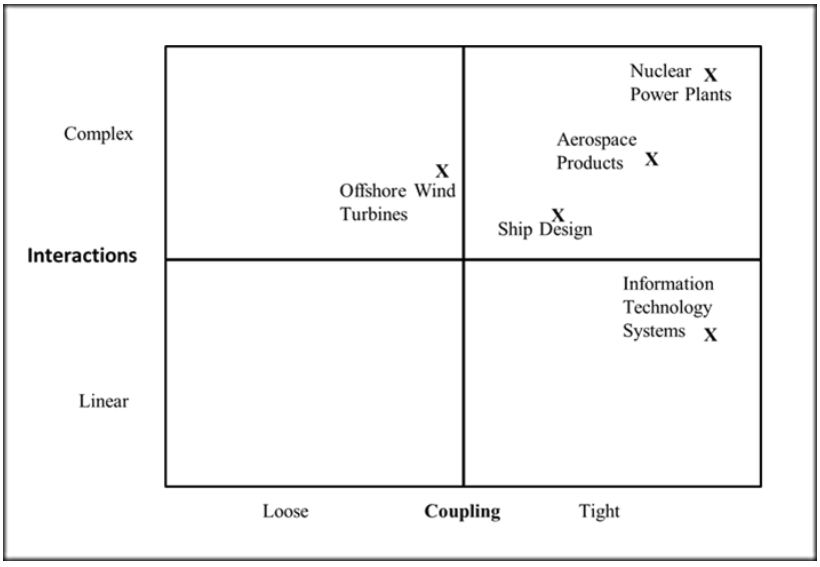

Figure 5. A diagram depicting the relationship between interactions and coupling for engineered systems. Adapted from [104].

In [105], Lopez-Ruiz et al. propose a complexity measurement based on a statistical description of systems known as statistical complexity to capture structure, organization, patterns, regularities, and symmetries in systems. This approach is to quantify complexity in term of randomness within a system. Systems are essentially message sources. For example, for a system whose message is "repeat 1 2," statistical complexity is low. On the other hand, for a system whose message is "choose at random from X, W, J, or S," statistical complexity is high. Statistical complexity is not easy to measure because it

requires interpretation of the system as a message source. In general, this measurement method is very difficult to apply. To apply this measurement in an engineered system, a systems engineer constructs the simplest possible model that represents the behavior of the system [81]. The amount of information required in an event to describe the behaviors of the simplest model is the measure of statistical complexity. Some scientists use this measurement approach to research crystals' atomic structure and neuron firing patterns [81].

Remington and Pollack [84] propose temporal complexity as the volatility or disruption caused by the external environment (market dynamics, social and political challenges, uncertainty in environmental regulations, and technical issues) during the course of the project. Consequently, this type of temporal complexity could affect project outcomes.

A summary of the research of these researchers appears in Table 5. It seems like some of these researchers characterize randomness as complexity and some researchers characterize randomness as not complexity. One characteristic of randomness is that there is no correlation between one sample and the next. Random behaviors of a system can be generated by either random noise processes produced internally or random external influences on system elements. The random external influence can be measured or known, and, therefore, the behavior of the system is predictable. Hence, random behavior is not complex. On the other hand, a complex system, by definition from Chapter I, may exhibit emergent, nonlinear, non-deterministic and/or chaotic behavior, and, thus, the dynamic behavior of a complex system can be either random or not random. In short, temporal complexity can be measured either based on the randomness within a system or the strength of coupled interactions between system components.

Table 5. Temporal complexity measures in engineered systems and engineering projects.

Year	Researchers	Temporal Complexity Metric
1988	Lloyd and Pagels [103]	Measure temporal complexity by calculating how many bits of information of the resources are required to describe the particular trajectories leading to the current state (i.e., randomness within a system).
1995	Lopez-Ruiz et al. [105]	Measure temporal complexity based on the randomness within a system (statistical complexity).
1999	Perrow [104]	Measure temporal complexity by the strength of coupled interactions between system components.
2002	Wolfram [11]	Measure temporal complexity based on everything that is not static, cyclic, or random.
2008	Remington and Pollack [84]	Measure temporal complexity in terms of external complexities associated with market volatility, funding mechanisms, and political and institutional complexities.

4. Technological Complexity

Technological complexity refers to the level of difficulty related to the conversion processes that transform inputs into outputs [14], [86]. For example, a drone control system has a sophisticated algorithm that analyzes and converts video signals to electrical signals in order to control the drone. The algorithm that performs the geo-location mapping as well as the transformation process of analog signals (videos) to digital signals (digitized electrical signals) is complex. To a certain extent, technological complexity can be described as the difficulty in creation of things [90]. It represents the level of difficulty concerning the transformation processes in engineered systems [90]. Lloyd proposes

complexity measurement in terms of time, energy, and/or dollars [90]. Lloyd's measurements include the following [90]:

- The amount of simulation (cost) involved to produce a prototype of a design.
- The length of a series of events (time) that leads to the system created.
- The least amount of information from a system's past behavior that is required to predict its future behavior.

This approach is useful in evaluating the speed of a particular algorithm in solving problems that involve searching, optimization, counting, and decision-making. An example of this measurement approach is the Google search-ranking algorithm [106], [107], which sorts through many webpages in a search index in a fraction of a second to find the most relevant and most useful results for what the user is looking for. The Google search-ranking algorithm uses all incoming links and checks the relevancy of the source websites so it can calculate a value for each link [106], [107]. The complexity of the ranking algorithm is that it has to process a large amount of data, and the processing mechanisms are constantly updated because of the dynamic nature of the World Wide Web [106].

Another example of applying this measurement approach is the particle-swarm optimization algorithm [108], [109], which solves a system optimization problem that is adaptive and emergent. A software engineer measures the amount of time it takes for the particle-swarm optimization algorithm to solve the multi-dimensional problem. This is a critical measurement for systems that process real-time data. In practice, this measurement approach is very difficult to apply to man-made systems whose evolution is unclear.

Bennett [110] proposes a complexity measure that computes the system's logical depth. Logical depth refers to the amount of computation or simulation time involved in producing a prototype of a design [110]. In other words, an object's logical depth is a measure of how difficult that object is to make [110]. A logical depth of an algorithm is a minimum size necessary for recreating a given piece of information [110]. To a certain extent, technological complexity can be interpreted as a logical depth of an algorithm. For example, one could conceptually use a Turing machine (i.e., a mathematical model of computation) and a software program to perform a computation [110]. The time required

for the optimum Turing machine to create the object is a measure of the complexity [110]. However, it is impossible to determine which Turing machine would be optimal, or how long it would take to run. In general, logically deep objects and algorithms take longer to compute, construct and run than to calculate, build and execute simple, flat structures of objects and algorithms [110]. In a man-made system that generates a design of any natural object of interest, this proposed measure of complexity is impractical. For example, a system that generates a design of a tree and its fruit yield would be difficult because of the importance of variables such as water supply and climate conditions.

Baccarini [14] describes technological complexity as the level of complexity in transforming inputs to outputs. According to Baccarini [14], elements of technological complexity include the characteristics of the project; the level of knowledge required for planning, executing, and managing the project; and the operational requirements [14]. For instance, for an engineering project that designs a computer chip, elements of technological complexity include the number of employees required for the project, the location of the project, the type of skill set needed, and the knowledge required to design computer chips [14].

Table 6 presents a summary of the work of these authors.

Table 6. Technological complexity measures in engineered systems and engineering projects.

Year	Authors	Technological Complexity Metric
1988	Bennett [110]	Measure technological complexity by the amount of computation or simulation time involved in producing a prototype of a design.
1996	Baccarini [14]	Measure technological complexity by "the number of operators required, location of the project, type of work skills needed, and type of technical expertise required."
2001	Lloyd [90]	Measure technological complexity based on time, energy, and/or dollars.

In summary, previous studies of complexity in systems design and development explained in this section suggest that there are many researchers who have presented many different possibilities for measuring complexities. This great quantity of complexity measures is reinforced by the number of pages it took to explain all of it. The list of complexity measurements in the four types of complexity may be insufficient and incomplete because complexity is multi-dimensional (e.g., complexity in the environment, in the system, in the design of the system as well as in the engineering of the system) and not precisely prescribed by a specific theory of complexity. Some measurements of complexity in engineered systems are difficult to perform because the measurement requires interpretation of the system as a message source and various observers also may disagree on the definition of depicting a system's regularity (e.g., statistical complexity and effective complexity). Some measures of complexity are impractical to use in actual systems, or may have major flaws because of the enormous amount of information and computation needed to generate a data model (e.g., algorithmic complexity). These limits open up the opportunity for this dissertation to develop practical measures of complexity. For instance, numerous studies on software complexity focus on the size of the code, control flow, data flow, the size of the project team, the number of interdependent programming tasks as well as other software metrics. This research can extend the complexity measurement to include requirements volatility and defect density as well as geographical distribution of project teams, and create a complexity profile for project risk assessment. In the next section, we review some measures of complexity in the literature related to nonlinearity, non-equilibrium, numerosity, connectedness, interdependence, diversity, and adaptivity.

5. Seven Building Blocks of Complexity

Both Page and Miller [13], [46] conclude that complexity arises in the "Interest in Between." "Interest in Between" refers to the spaces between simple and strategic behaviors. These spaces lie between what engineers already know and engineers need to know. Page observes that a certain mix of values for attributes (nonlinearity, non-equilibrium character, numerosity, connectedness, interdependence, diversity, and adaptivity) may lead to complexity [13]. A different mix may lead to a different kind of

system [13]. This is an important observation because models that study complex systems need to account for the "Interest in Between" phenomenon. When a system bears none of the Page's seven attributes of complexity, it is essentially in an inert state, and complex behavior is not possible. Furthermore, it seems that no single source but rather a combination of the seven attributes appears to increase complexity. This realization is important to this research because of the need for assessing the risk of a complex project. Let us review the complexity measures related to the seven building blocks of complexity.

(1) Nonlinearity

As described in Chapter I, nonlinearity is a necessary condition for complexity [29]. Nonlinear system behaviors are complex. We explained in Chapter I the "bullwhip" effect [39] in which we have unproportioned fluctuations of orders in the supply chain. This phenomenon illustrates one example of how a nonlinear system behavior is indicative of a complex system. In [111], Allgöwer proposes a nonlinearity measure based on two approaches. One approach is to calculate the lower bound of Fourier coefficients of the system's output. The second approach is a numerical computation of the nonlinearity measure (a nonlinear function describing the behaviors of a system) using convex optimization. Both approaches are difficult to apply to man-made systems.

By analogy, in the complex forms of nature, multimodality provides for rugged landscapes and numerous possible bifurcations [112]. Hence, a systems engineer might be able to measure nonlinearity that matters to complexity in an engineered system by estimating the multimodality of a nonlinear function. An approach to estimating multimodality is to use the maximum exponent in a power series expansion of the nonlinear function that describes the behaviors of the system [29]. However, when coefficients in a power series expansion become sufficiently small, one might terminate the expansion. Unfortunately, determining multimodality of an arbitrary function does not appear to always yield useful solutions. As a possible alternative, a qualitative approach using descriptive analysis and subject-matter experts' experience to rate the complexity level of the parts' behavior may be a reasonable method to measure nonlinearity. Systems engineers

can analyze the behaviors of the parts to predict the system's behavior. Nevertheless, measurement of nonlinearity is still an active area of research.

(2) Non-equilibrium

As stated in Chapter I, a non-equilibrium system is a system that dissipates energy persistently regardless the amount of energy supplied from the external systems. As discussed in Chapter I, Chaisson's work [19] shows that as a system increases in the flow of energy, the more capability that system has to reduce entropy, and thus, the more complex is the system. Intuitively, the flow of information links to the dynamics of the system, and dynamics are related to either information processing, work performed, or the flow of energy.

In [35], Clark and Jacques propose that one possible way to determine complexity in dynamic systems is to observe changes in kinetic energy. This measurement allows systems engineers to understand system risks and analyze the dynamics of the system that might be the cause of the failure of the system [35]. This work is still in its infancy and requires further study.

In engineering projects, several authors have measured the flow of information of a system [113], [114], [115], or the flow of information of the business processes [92]. However, some of these measurements require production schedules (e.g., orders and deliveries) and supply chain delivery schedules, and therefore are not practical to implement in the early stages of a project. In short, measurement of non-equilibrium is still an active area of research.

(3) Numerosity

In Chapter I, we discussed numerosity in some detail as it related to size measures and the number of observable behaviors in a system. For example, an F-35 fighter jet contains many systems and subsystems. The interactions of systems and subsystems can produce emergent behaviors. Thus, numerosity appears to be a necessary condition for complexity. Meyer et al. [116] present a simple way to calculate numerosity in a system by multiplying together the number of parts (N_p), the number of types of parts (N_t), and the

number of interfaces of each of the parts (N_i) and then taking the cube root of the product as shown in Equation 2.1.

$$\text{Numerosity} = \sqrt[3]{(Np)(Nt)(Ni)} \tag{2.1}$$

This method of measuring numerosity requires a complete model (down to the level of individual parts) of a system. Furthermore, this method assumes that all three factors (the part, the type of parts, and the interface of the part) contribute equally to the level of complexity in a system, which is not always the case in actual systems.

In [24], Valerdi proposes a COSYSMO model to estimate "the time and effort associated with accomplishing the system engineering tasks" in large-scale systems. Valerdi found various parametric drivers that could affect the cost of a project [24]. The effort and duration of a project required for the completion of project tasks are often expressed in terms of person-month. COSYSMO's size drivers of systems engineering are the number of requirements, use cases, interfaces, and algorithms that are applicable to systems containing both hardware and software [24]. Table 7 presents the sources of information for the four size drivers of systems engineering in COSYSMO.

Table 7. Size drivers of systems engineering and corresponding sources. Adapted from [24].

Size Driver of Systems Engineering	Source
1) Number of system requirements	Counted from the document of system requirements specification
2) Number of major interface	Counted from the interface control document
3) Number of critical algorithm	Counted from system requirements specification or system design description document
4) Number of operational scenarios	Counted from use cases or test cases

Early in the system development life cycle, these sources of size drivers may not be available yet. As an alternative, program managers can obtain or derive data for these size drivers from previous acquisition programs. From the SE perspective, the functional size of a system is estimated by the weighted sum of these drivers [24]. Table 8 presents the relative weights for each level of size drivers. These weights are based on a survey data from systems engineering experts in various application domains. For instance, as shown in Table 8, a "difficult" system requirement has a relative weight of 5hereas, a "nominal" operational scenario has a relative weight of 22.8.

Table 8. Relative weights for size drivers of systems engineering based on a survey data from experts in the field. Source: [24].

Size Driver of Systems Engineering	Easy	Nominal	Difficult
1) System Requirement	0.5	1	5
2) Major Interfaces	1.7	4.3	9.8
3) Critical Algorithms	3.4	6.5	18.2
4) Operational Scenario	9.8	22.8	47.4

Factors such as complexity, volatility, and reuse can be used to adjust the size drivers as shown in Table 8 [24]. However, Valerdi [24] only chooses complexity to characterize the size drivers of systems engineering and uses a 6-level scale (very low, low, nominal, high, very high, and extra high), as shown in Table 9.

Table 9. Rating values for cost drivers of systems engineering. Source: [24].

	Very Low	Low	Nominal	High	Very High	Extra High	*EMR*
Requirements Understanding	1.87	1.37	1.00	0.77	0.60		*3.12*
Architecture Understanding	1.64	1.28	1.00	0.81	0.65		*2.52*
Level of Service Requirements	0.62	0.79	1.00	1.36	1.85		*2.98*
Migration Complexity			1.00	1.25	1.55	1.93	*1.93*
Technology Risk	0.67	0.82	1.00	1.32	1.75		*2.61*
Documentation	0.78	0.88	1.00	1.13	1.28		*1.64*
# and diversity of installations/platforms			1.00	1.23	1.52	1.87	*1.87*
# of recursive levels in the design	0.76	0.87	1.00	1.21	1.47		*1.93*
Stakeholder team cohesion	1.50	1.22	1.00	0.81	0.65		*2.31*
Personnel/team capability	1.50	1.22	1.00	0.81	0.65		*2.31*
Personnel experience/continuity	1.48	1.22	1.00	0.82	0.67		*2.21*
Process capability	1.47	1.21	1.00	0.88	0.77	0.68	*2.16*
Multisite coordination	1.39	1.18	1.00	0.90	0.80	0.72	*1.93*
Tool support	1.39	1.18	1.00	0.85	0.72		*1.93*

COSYSMO [24] provides an estimation of the functional size of a system in terms of the number of person-month necessary to develop the SoI. In general, we need more personnel to develop large systems [24]. Equation (2.2) shows the mathematical form of the COSYSMO model [24].

$$PM = (A)(Size)^{E}\left(\prod_{i=1}^{n} EM_i\right) \tag{2.2}$$

PM denotes effort "in person-month" [24] for a formal schedule. "*A* denotes a calibration constant" calculated by "historical project data; a typical value is 0.325 for a software development project" [24]. "*Size* denotes the summation of the weighted four size drivers" [24]. *E* denotes efficiency factor "where the default is 1 and *n* denotes the number of cost drivers. EM_i denotes effort weight factor for the i^{th} cost driver where the nominal value is 1" [24].

In estimation of the cost of software, "lines of code (LOC), function points (FP), and application points (AP)" are common measures of size of the software [24]. The COSYSMO have "adjustment factors to estimate the size of software for different

languages running on different platforms." Nevertheless, some limitations of COSYSMO include the following [24]:

- The range of operation of COSYSMO is for application domains in the military/defense and space systems fields because the model was validated mostly from defense and space industry data.
- COSYSMO ignores the effects of volatility and reusability in size drivers.
- COSYSMO lacks rules and standards regarding the counting of requirements.
- COSYSMO does not include the effect of systems engineering organizations on systems engineering efforts.

Despite these limitations, COSYSMO identifies "size and cost drivers" for SE projects, which can serve as a risk management tool for SE organizations [24]. Ultimately, after performing COSYSMO estimation, program managers need to go through a common-sense test to determine whether the system warrants the size and cost of systems engineering.

(4) Connectedness

In engineered systems, we measure connectedness by counting the number of interactions between systems. The types of interactions include the exchange of energy, information, money, and material. In general, as the number of links (e.g., dependences, relationships) in a network increases, the connectedness of the system increases, and at the same time, so does the complexity of the system [117].

In engineered systems, we measure connectedness by calculating the number of entities with which an entity can directly connect or interact. Entities include systems, people, computers, and machines. Entities may be connected in physical, informational, geographical, and/or virtual environments. Two connected entities may, or may not, interact directly. They may exchange information via networks.

In [118], Bailey et al. propose an index measure of social connectedness based on the relative frequency of friendship connections (e.g., patterns of relationships) in large social networks (i.e., Facebook, LinkedIn). This study shows that the geographic distance contributes a decrease to the intensity of friendship connections [118] (e.g., the Allen curve [119] phenomenon). It also demonstrates that "social interaction correlates with social and economic activity across regions" [118] (e.g., engagements and interactions between people to exchange information or money in a clustered network).

In addition, Greenwood-Nimmo et al. [120] develop a technique to measure macroeconomic connectedness by studying financial connectedness in a global financial system. This technique relies on the topology structure of clustered networks in global systems to identify and analyze systemic risk, which, in turn, can be used as a risk mitigation tool for global financial systems. Both Bailey et al. [118] and Greenwood-Nimmo et al. [120] identify and analyze the number of observable behaviors (i.e., patterns), which is an indirect measure of connectedness in a clustered network.

From a systems engineering perspective, modern engineered systems often have internal networks that connect components and external networks that connect the system to other systems. Information processing supports connectedness by facilitating many possible implementations of networks.

(5) Interdependence

Interdependence emerges due to the flow of processes [28]. In engineered systems, interdependence is the influence of the behavior of one entity upon the behavior of another, and vice versa [28]. Interdependence can occur when entities are connected either in physical, informational, and/or virtual environments. Information processing supports interdependence because any message sent from one node to another requires transmission, reception, and subsequent action for any dependence to occur [28]. In systems engineering, interdependence is an informational, control, or resource relationship such that one organizational unit depends on another organizational unit [27].

Danziger et al. [121] describe interdependence as the interactions between networks via connectivity and dependency links in the context of critical infrastructure. When links

connect nodes within the same network, we have connectivity in the network [121]. Danziger et al. state: “Links that depend on other links require support from another network” [121]. Although the dependent node is connected to its own network, “the dependent node will fail if the supporting node fails” [121]. When two networks support each other for connectivity, we call these two networks interdependent networks [121].

Amaral et al. [122] propose an interrelatedness measurement between the different parts or sectors of an economy. This measure reflects two aspects of complexity [122]: (1) every part of a system has connections to other parts of the system (i.e., more connections would increase complexity); (2) the behavior of each part of the system in relation to both the internal connections among the elements of that part and external connections with other parts of the system.

In [123], Domercant et al. propose a complexity measure of military SoS architecture by capturing three aspects of system complexity: (1) “the distribution of functionality among constituent systems within the architecture,” (2) the interfaces that connect these systems, and (3) “the coordination of numerous functions to fulfill a capability.” In essence, Domercant et al. propose a complexity measure of SoS architecture that captures the degree of interdependence among functions, tasks, and activities as well as the flow of information and data through the interfaces.

In an enterprise system, interdependence is the degree to which one unit’s actions control or influence another unit’s actions and outcomes, and vice versa [28]. In [28], Giachetti proposes a model to characterize the strength of interdependence between the subsystems of an enterprise. He views interdependence as occurring between tasks and exhibiting asymmetric relationship (e.g., task A can be dependent on task B but the reverse may not be true) [28]. Giachetti’s model captures the attributes of workflows in an organizational structure and measures these attributes based on frequency, importance, and delay [28]. Based on Giachetti’s interdependence definition in an enterprise system [28], it appears that one way to measure interdependence is to develop a network model using organizational units as nodes and the frequency and the level of importance of the flow of information between each organizational unit as links to measure interdependence.

(6) Diversity

A system's diversity stems from the number of different types of elements it comprises. Systems engineers might observe that diversity has several relationships with complexity. First, diversity in types is often associated with complexity. Second, diversity in compositions is also often associated with complexity. Third, variation within a type seldom results in complexity because differences in a type are describable by a numerical value. Diversity of type and variation within a type as described by the law of requisite variety [124] generally improve robustness, which is the ability to maintain functionality despite a disturbance. For example, if a system has a number of different types of system software, then the system has more functions available to exploit for responding to a disturbance. If a system can support a number of different screen sizes, then the system likely finds one that will fit certain devices to support a number of users. Hence, diversity usually improves performance and generally increases behavior [29]. Furthermore, complex systems tend to have diversity [29].

However, it is possible to have too much diversity. For instance, if a system has 200 different kinds of parts, then there are about 20,000 possible ways to combine two parts into a single part for reducing the number of parts in a system. If a system has different entities, then each entity in the system has more ways to change. If different kinds of entities present, then they have more ways to combine.

In [125], Broekel suggests that the complexity of technologies can be measured by the diversity of components within the network structure of a system. He defines technologies as combinations of components connected in various types of network topologies [125]. From his technology complexity study, Broekel [125] concluded that as "the range of functions" of a system increase over time, technologies become more complex. In addition, Broekel [125] also concluded that complexity of technologies are associated with "large research and development (R&D) efforts and intensive collaborative R&D activities." Broekel's model [125] of complexity of technologies reflects characteristics of both numerosity and diversity.

In Chapter I, we provided an example that different types of job positions in interdependent organizational units are likely to foster differences in job assignments and outcomes, and, therefore, increased complexity. We proposed modeling diversity by mapping out different types of job positions of organizations involved in the project, and then counting the total number of different position types. In general, a complex engineering project has different types of organizational units. Each organizational unit generally has different types of job positions to perform some tasks. This approach of modeling diversity reflects both the variation and attribute measurements.

(7) Adaptivity

Adaptivity refers to a system's ability to change to accommodate the new environment. It is a source of robustness. Complex adaptive systems (CAS) are systems that continuously interact with their environment [126]. The interaction can be the transfer of information, energy, or material into or out of the system boundary. CAS have a history or memory as they evolve, and they show nonlinear behaviors. Examples of CAS are virtual reality (VR) systems, augmented reality (AR) systems, and networks of "artificial intelligence systems" [126].

In engineered systems, adaptation is relatively fast, and it requires computation (information processing) [29]. However, in a natural world, evolution is the primary mechanism for adaptation, which is not fast and does not require computation. Evolution is a process of moderate change from a simpler state to a more complex state [29]. For example, mountain ranges rise up and erode down gradually over time. In systems engineering, we have systems that are made from technologies, and those technologies evolve over time because humans direct this evolution (e.g., an F-18 fighter jet has better technology than an F-14). Adaptivity is difficult to quantify because it potentially involves every element of a system and every activity the system can perform. One possible way to determine the adaptivity in a system is using a subjective scale similar to a Likert scale [32]. Creating a subjective scale for relative rankings is not new in the scientific community. This approach of measuring adaptivity has not been proven through validation in empirical research studies.

Siddiqa et al. [127] review a form of complex adaptive networks called the Social Internet of Things (SIoT) in which nodes are interconnected and act as intelligent agents. These agents can accept changes and adapt to their environment for its survival. The idea of adaptivity in the SIoT can possibly be measured by the degree of resiliency of network connectivity and network traffic load. This approach of measuring adaptivity has not been proven through validation in empirical research studies.

In [128], Tamersoy et al. define a requirements model for adaptive multi-organizational systems based on non-functional requirements (i.e., goals, roles, and organizations). The model of adaptivity relies on the dependencies between organizations and between roles. Adaptive systems consist of multiple interacting organizations. Each organization has multiple interacting roles, and each role has multiple goals. The degree of adaptivity of a system is determined by the patterns of interacting organizations and roles. Nevertheless, this approach of measuring adaptivity in a system serves as a proof of concept. Though, more studies might be desirable.

In Chapter I, we discussed the architecture adaptability index (AAI) and the software adaptability index (SAI) proposed by Subramanian and Chung [33] for measuring software adaptivity. Both adaptivity indexes appear evolvable and scalable. For example, in the beginning of a project, the project manager can calculate the SAI based on the module architecture. At the end of the project, the project manager can measure the SAI based on the final code. Hence, the project manager can verify that a certain software has a certain adequate level of adaptivity. The drawbacks are that AAI and SAI are limited to measure adaptivity for non-functional requirements, and software can only satisfy non-functional requirements within acceptable limits. Nevertheless, Subramanian et al.'s approach can serve as a good starting point for measuring adaptivity in software. In short, it appears some research has been conducted on the application of CAS to engineering systems.

In conclusion, nonlinearity, non-equilibrium, numerosity, connectedness, and interdependence are characteristic of all systems. Diversity and adaptivity are the result of complexity, and they are present in complex systems but not in ordinary systems. The authors of [24], [116], [125] point out that most size measures do not appear directly related

to complexity. Instead, they may be better indicators of complicatedness. However, because numerosity is a factor in complexity, size must play some role in the degree of complexity [24], [125]. The combination of numerosity and diversity in a system appears to increase complexity. In addition, several authors such as [28], [118], [120], [122] show that the quantity of interactions is related to numerosity, connectedness, and interdependence. The quantity of interactions potentially increases as the number of parts increases and/or the number of connections between parts increases. Furthermore, as the number of connections increases, interdependence increases [28], [122], [123]. However, the quantity of interactions does not depend on the degree of diversity, and it ignores adaptivity [29].

Research in methods to measure complexity will play a major role in developing solutions to many complex engineered systems. There exists a gap between what is available in the DOD systems acquisition programs today and the complexity measurement models that could be used in the early phases of those military acquisition products [129]–[131]. The goal of this dissertation is to narrow this gap. In the next two sections, we first discuss software complexity measurement from previous works and then review the literature on project complexity measurement. Finally, we identify some gaps in the literature concerning complexity measurement and propose some ideas to address these gaps. Ultimately, this dissertation aims to propose a complexity measurement model that can assess project risk and aid the development team to reduce project complexity.

6. Software Complexity Measurement

Software complexity involves the structure and content of the software. Complexity in software commonly contributes to software error. We measure the complexity of software systems by software size, control flow, data flow, and other factors influenced by the specific programming task and the experience of the programmer [132], [133]. In general, the cost of software development and maintenance can be estimated by software size and software complexity. To reduce and contain the software development cost and software defects, software engineers and project managers must find relationships between programming tasks (i.e., coding, troubleshooting, and unit testing or changing the

software), development time, and program maintenance [133]. Project managers and software engineers use measures of software complexity for cost projection, labor allocation, and program evaluation [133]. In general, software complexity measures have two categories: design complexity and code complexity [132]. Software engineers assess software design and code complexity from program size, program control flow (i.e., logic structures in a program) and program data flow (i.e., data dependency among modules) [132]. These measures are the physical activities in the software development life cycle, and they predict software reliability, portability, and maintainability [132]. In an attempt to understand the sources of software complexity, researchers consider the following questions [132], [133]:

- How much does it cost to maintain complex software?
- What are effective techniques to measure software complexity?
- What metrics are useful and effective for identifying software elements and patterns that incur avoidable complexity?

Early in a development cycle, project teams can gauge source code complexity to target potential problem areas [133]. The following sections identify developments in measures of software complexity.

a. Lines of Code (LOC):

Lines of code (LOC) is a very common measure used to quantify software complexity. It counts the number of lines of source code, which indicates program size [134], [135]. The advantages of the LOC measure are as follows [134], [135]:

- It is simple and easy to count.
- It is independent of program language.
- It is easy to understand.
- It can automate the counting process.

The disadvantages of the LOC measure are as follows [134], [135]:

- It does not account for the intelligence content and the layout of the code.
- It ignores the complexity from the program control flow.
- It has no counting standard because different tools measure different things (e.g., commented lines, blank lines, and program statements) and provide different results.

LOC is widely used as an estimate of program size [134]. In addition, it can be used with other code metrics to formulate more comprehensive complexity determination [134].

b. McCabe Cyclomatic Complexity Measures

In 1976, McCabe [136] suggested cyclomatic complexity measures to determine software complexity. He defined cyclomatic complexity as the number of linear free paths within source code, which is simply the number of decision points plus one [136]. Cyclomatic complexity is used to measure the relative complexity of different builds of the software. Cyclomatic complexity depends entirely on the structure of software's control flow graph. Industry studies indicate that a high cyclomatic complexity will lead to a high probability of errors [136]. In addition, many industry studies have shown that the number of tests required for a software method indicates that method's cyclomatic complexity [136]. Thus, in attempting to increase overall reliability, many organizations limit the cyclomatic complexity of their software methods.

McCabe proposed to limit cyclomatic complexity to 10 [136]. Limiting complexity at all stages of software development helps to avoid risks associated with high complexity software [136]. Overly complex methods have the following effects [136]:

- They are prone to error.
- They are hard to understand.
- They are hard to test.
- They are hard to modify.

In practice, software systems engineers use tools for software metrics to analyze and calculate cyclomatic complexity. For example, Java code developers use an Eclipse Metrics plugin to compute cyclomatic complexity [137]. C# developers use ReSharper plugin to calculate cyclomatic complexity [138].

Advantages of McCabe cyclomatic complexity are as follows [135]:

- It can be applied early in a software's life cycle (software design phase).
- It provides relative complexity of various software design patterns.
- It is easy to apply.
- It guides the testing process and maintenance activities by limiting the program logic during development.

Some disadvantages of McCabe cyclomatic complexity include the following [135]:

- The control flow graph must be available, which does not occur until late into the software design phase.
- It may provide a misleading figure because many of McCabe's tools measure different variables and can yield different results.
- Interpreting the results of cyclomatic complexity is a challenge because software methods with a high cyclomatic complexity score can also be easy to understand, test, and maintain, and vice versa. In other words, cyclomatic complexity does not distinguish the complexities of various kinds of control flow.
- It ignores both the complexity from the program's data flow and complexity added at the nesting levels.

c. Halstead's Complexity Metrics

In 1977, Halstead introduced metrics for measuring complexity of software [139]. Halstead's metrics measure the properties of software and the relationships between them. These metrics provide an estimate of the program's vocabulary, size of the program, volume of the program, difficulty to develop the program, errors generated by executing the program, and effort required to create the program as well as time needed to develop source code. The McCabe Software Company suggested limiting the size of the program N to 300, volume V to 1,500, difficult D to 30, effort E to 300, and estimated software defects B to 0.6 [140].

Advantages of Halstead's metrics are as follows [135], [139]:

- Any programming language can apply the Halstead model.
- It does not need in-depth analysis of programming structure.
- It predicts error rate, maintenance effort, and overall quality of programs.

Some disadvantages of Halstead's metric include the following [135], [139]:

- It needs a completed code to compute the metrics. Thus, it has little use as a predictive estimating model.
- Because of the absence of any standard of counting rules in a program, it is difficult to define and count a distinct number of operators and operands in a program.
- It does not specify what the level of the program that makes the program complex.
- There are flaws in the assumptions used to derive Halstead's formulas.

d. The Complexity Metric of Henry and Kafura

In 1981, Henry and Kafura proposed a complexity metric based on the procedure length and "the flow of information into procedures and out of procedures" [141], [142].

Henry used “the UNIX system to validate his complexity metric.” He suggested that “the complexity of a component” can be an indicator of “potentially faulty components” of a system [142].

Advantages of Henry and Kafura’s metric are as follows [142]:

- It can be applied during the design phase in a software life cycle.
- It accounts for data-driven complexity.

Disadvantages of Henry and Kafura’s metric are as follows [142]:

- If a procedure has no external interactions, it will have a complexity value of zero.
- This method cannot apply for black box components when the component’s source code is unavailable.
- For complicated and complex method calls, it is very easy to miss the flow of control relations.
- This method requires some techniques for thorough information flow analysis of the software method.

e. The Complexity Metrics of Chidamber and Kemerer

In 1993, Chidamber and Kemerer proposed six complexity metrics for object-oriented software design [143]. They are as follows:

(1) Weighted methods per class (WMC)

This measure [143] calculates the sum of the complexity of all the methods identified in the class $WMC = \sum_{i=0}^{n} C_i$ where n is the totality of methods identified in a class and C_i is the class complexity. Because the high value of this metric indicates a high degree of complexity in the class, it requires more testing effort [143]. In software development, a short method is preferred because it is easy to read and can reduce program complexity [143].

(2) Depth of inheritance tree (DIT)

This measure [143] computes the maximum number of steps from the root class tree to the class node. A high DIT value indicates a high degree of complexity in design, but it also represents an excellent opportunity for reuse of inherited methods [143].

(3) Number of children (NOC)

This measure [143] provides the sum of immediate successors of the base class. A high NOC value indicates poor design needing more testing effort, but it also represents a huge potential for reuse of base class [143].

(4) Response for class (RFC)

This measure [143] offers the total invoked methods from a message received by an object of the class. A high RFC value represents a high degree of complexity in the class and, therefore, it requires additional testing effort. A high RFC value is also associated with software defects [144]. Thus, in software development, a low RFC value is preferred because it indicates lower complexity and defects [144].

(5) Coupling between object classes (CBO)

This metric [143] measures the level of coupling between object classes in the code. When an object class calls the methods of another object class or vice versa, we have a coupling between two classes. When coupling between classes is high, programmers need additional time to analyze and alter the codes. A high CBO value indicates poor design, difficulty in understanding interaction between classes, and less reuse of classes [143]. Thus, the software requires more maintenance effort. Sahraoui et al. recommended keeping CBO values below 14 [145].

(6) Lack of cohesion method (LCOM)

This measure [143] identifies the cohesion of a class. It is the sum of disjoint sets of methods in a class. A low LCOM value indicates a high degree of cohesion in a class and, therefore, implies simplicity and high reusability of the class [143]. A high LCOM value indicates potential software defects [144].

Advantages of Chidamber and Kemerer's metric are as follows [143]:

- It supports object-oriented programs.
- It is language independent.
- Some metrics highly correlate to software defects, which are useful for software developers and testers.

Some disadvantages of Chidamber and Kemerer's metric include the following [143]:

- The *WMC* metric cannot apply for black box components when the component's source code is unavailable. Thus, it cannot apply during the design phase in a software life cycle.
- Relationships between metrics (e.g., DIT, NOC, and CBO) and defects are incoherent.

f. Function Point

In 1979, Albrecht at IBM introduced the function point approach to estimating software size [96]. Function point is an approach to estimating software size (e.g., lines of code) [97], and as a size measure, it is an estimate of complexity. To compute function points, software engineers first determine the system's features and functionality based on five parameters [148]: external inputs, external outputs, internal logical files, external interface files, and external inquiries. Each parameter of a function point has three categories: simple, average, or complex [148].

To compute the raw function points, software engineers adjust each parameter of a function point by a weight number shown in Table 10. Each function point has an associated level of complexity. For example, a number is a simple user-output type, while an image is a complex user-output type. In sum, function point analysis captures certain aspects of software complexity including distributed data processing, data communication, and complex processing.

Table 10. Weight scale for function point parameters. Adapted from [97], [148].

Function Point Parameter	Complex	Average	Simple
User Inputs	6	4	3
Outputs send to users	7	5	4
Internal logical files	15	10	7
External interfaces	10	7	5
External inquiries	6	4	3

Other software metrics for assessing software complexity include counting the link-time dependencies in an application and measuring an application's balance between the degree of abstraction and stability of related classes in object-oriented programming [149], [150].

A summary of measures of software complexity appears in Table 11.

Table 11. Measures of software complexity.

Metric	**Software Complexity Measurement**
Lines of code (LOC) [134], [135]	This measurement is easy to count but not available early in life cycle.
Cyclomatic complexity (McCabe) [136]	This measurement is well studied, easy to compute, internal to a module, and strongly correlated with effort and propensity for defect.
Halstead complexity [139]	Similar to McCabe's cyclomatic complexity, this measurement counts the number of operators and operands in the software.
Henry and Kafura metric [141], [142]	This measurement counts the number of relations among modules and is easy to compute.

Metric	Software Complexity Measurement
Chidamber and Kemerer metric [143]	This measurement supports object-oriented programs and highly correlates to software defect.
Function point [96]	This measurement is available prior to code and is easy to understand.

7. Project Complexity

As projects have become more complex, it is imperative for project managers to understand project complexity. In general, complexity affects the schedule, cost, and performance of a project. The higher the project complexity, the more time it takes to complete the project and, therefore, the higher the costs to achieve project outcomes [148]. To create a project plan of a software program, the project manager needs to determine the project cost by estimating the number of personnel required for the project. To estimate the project cost, the project manager needs to estimate the average yearly salary of a software developer. The Project Management Institute (PMI) defines four project sizes [151]: (1) small projects of less than $5 million, (2) medium projects between $5 million to $50 million, (3) large projects between $50 million to $500 million, and (4) mega projects above $500 million. In the computer software industry, the average yearly salary of a software developer is $97,288 [152]. Adding 50% of cost for company-paid benefits and taxes [153], a software developer's yearly total cost in the U.S. is $145,932. Hence, for a project cost of $5 million per year, the project manager can hire 34 software developers ($5 million / $145,932). We will use $145,932 yearly cost for a software developer as an industry standard in the software engineering projects.

The project complexity influences the selection of a project's organizational structure and subject-matter experts as well as experienced management personnel [149], [150], [154]. Through a review of the literature, we find that the work of several researchers can be grouped into the four approaches to understanding overall project complexity [14], [84], [155], [156]: (1) organizational complexity, (2) requirements complexity, (3) temporal complexity, and (4) technological, social, cultural, and task complexity. Let us

discuss each of the four approaches to examine their merits for measuring project complexity.

a. Organizational Complexity

Baccarini (1996) defines project complexity as the integration of an organizational structure and project management processes by coordination, communication, and control [14]. He refers to organizational complexity as a degree of difficulty that depends on the number of project elements (e.g., stakeholders, engineering disciplines, project size, number of employees, and business processes) and the relative interdependence of these elements [14]. In other words, organizational complexity relates to the organizational structure of the project team and the system development process [14]. Baccarini describes organizational complexity as the engagement of a virtual and logical multi-organizational structure to manage the project [14].

In [157], Wood and Ashton indicate that relationship difficulties between the project teams lead to poor dissemination of information and resulting in incoherent stream of goals at the outset of a project. Poor channels of communication, poor production, and use of information contribute to organizational complexity [157]. Wood and Ashton conclude that organizational complexity has a major impact upon the project complexity [157].

Schwandt [158] hypothesizes that "organizational complexity and organizational performance" (i.e., efficiency and value of shareholder) have "a bathtub-shaped relationship." The organizational performance will peak at a manageable level of organizational complexity and then the organizational performance will drop off quickly as an increase in organizational complexity [158]. He defines four drivers of complexity in his model [158]: (1) diversity, (2) ambiguity, (3) interdependence, and (4) fast flux. When the values of these drivers are increase, we have higher complexity [158].

Examples of organizational diversity measures include the number of institutions with which the organization interacts, the amount of budgetary resources, the number of stakeholders, and the amount of technical knowledge necessary to interact with team members [158]. Examples of measures of organizational ambiguity are the number of

subsidiaries, the extent of role variety, the number of the communications among organizations, and the level of efficiency of the overall business processes [158]. The extent of variety of roles are computed by the number of different types of job positions in an organization [158]. A role variety matrix contains job titles and responsibilities. An organizational chart provides the information needed for this measurement. Measures of organizational interdependence consist of the number of subsidiaries and organizational structure [158]. Measures of organizational fast influx include the proportion of new employees and the number and volume of mergers and acquisitions.

According to Schwandt [158], a structural size of an organization and its interdependencies are closely related because the growing number of elements in an organization will result in an increase in the number of relationships. Hence, the structural size of an organization reflects the organization's complexity [158]. In the reliability evaluation of Schwandt's organizational complexity measurement model, the weight factor of a size measure is 0.61 [158]. This means that the structural size of an organization has a significant impact on organizational complexity. One way to interpret this size's weight factor is by relating it to organizational complexity based on the structural size of a project team. For instance, if a project team consists of 9 organizational units, then the weight factor of organizational complexity of the project is 5.49 (9 times 0.61). We will use this size's weight factor as a reference value for comparison of organizational complexity in the engineering projects.

b. Requirements Complexity

Software developers must make changes in the code to reflect changes in requirements. According to Stark et al. [155], requirements volatility is positively correlated with the increase in the size of the project as well as with schedule duration, and accounts for 20% of the total maintenance effort. Requirements volatility is an indicator of the degree of stability of requirements. According to Stark et al. [155], requirements volatility is the number of changes to requirements (new requirements, modifications of requirements, and deletions of requirements) over a certain period divided by the number of requirements at the start of the project. Hence, understanding and analyzing

requirements volatility and its impact during the software development life cycle are important for managing project risk. In general, high requirements volatility contributes to low quality and low success rates of the delivery of software to customers [155]. Project managers and system developers are interested in requirements volatility for project management, risk assessment, and a potential assessment of project complexity [155].

Malaiya and Denton [159] show that high requirements volatility will cause the resulting software to have a higher defect density, especially when the requirement changes occur close to the software release date. High volatility of requirement changes reflects poor understanding of the environment and system, which can introduce problems in all areas of the system development such as software defects or poor quality of products [159]. Defect density is an important measurement of software quality and is often used as a software acceptance criterion. In the defect density measurements, the best practice in the software industry is to keep the number of defects per KLOC less than 3 for reducing the risk of requirements creep [160].

Pfahl and Lebsanft [161] demonstrate in their simulation that requirements volatility has a significant impact on the development effort (i.e., overall cost of the project) and project duration. Ultimately, requirements volatility represents risk to the success and completion of a project.

Parsons-Hann and Liu (2005) conclude that requirements complexity contributes to project failure [162]. Furthermore, Parsons-Hann and Liu identify five factors influencing the complexity of requirements [162]: (1) the time spent on the project, (2) the heterogeneity of the organization, (3) the skill of project members, (4) the number and location of stakeholders, and (5) the project resources. Let's explain each factor in order.

Because of project deadlines, a project manager often needs to estimate how long it will take to fulfill a certain requirement in a project and to complete an entire project. When the team estimates that a longer schedule is needed to complete the project based on the initial plan, then that project should be considered as a "high risk" project.

To stay competitive in the industry and to maintain good communication channels, many projects will have partnerships with geographically dispersed organizations. This can

be challenging for the ongoing relationships with business partners at various sites because of the differences in time zones, business and culture norms, native languages, and technologies used for communication. Thus, geographic barriers can affect the requirements complexity. The heterogeneity of the organization can contribute to the complexity of the requirements because of ongoing collaboration with stakeholders [162]. Azim's research [163] finds that 56% of the respondents cited that geographical distribution of project teams contributes to project complexity. One study from Waber et al. [164] shows that engineers who worked at the same building were 20% more likely to stay in touch digitally than those who worked at different sites. When a project team needed to collaborate, team members in the same office e-mailed each other four times as frequently as colleagues in different locations. Consequently, the increase in collaboration with project members lead to 32% faster of completing the project [164]. We will use Waber's study [164] as a criterion for comparing the measure of geographical distribution of teams in the engineering projects.

Most projects will have a range of skilled project members ranging from new graduates and young systems engineers to the project manager who have a number of years' experience working on similar projects. This diversity of skill level can affect the complexity of user requirements in terms of understanding of the system requirements and whether the project member has worked on a similar project before [162].

When managing a project in a virtual organization, coordinating and obtaining requirements from stakeholders at various sites is very difficult [162]. A stakeholder refers to any individual or group of employees who can influence or be influenced by an organization's activities [162]. The number and location of stakeholders can contribute to requirements complexity because of the difficulty of generating the requirements from numerous different stakeholders [162].

In a 1995 CHAOS report [165], 10.6% of the companies interviewed stated that the lack of project resources is the third biggest reason contributing to project failure. Project resources refer to money, employees, and skills required to meet all project objectives accurately and on time.

Nurmuliani et al. (2006) show that the major causes of requirements volatility are "changes in the needs of the customer, an increase understanding of the product by the developers, and changes in the policy of the organization" [166]. They show that requirements volatility is very high at the end of a requirement analysis phase [166]. They find that new requirements added late in the development life cycle require more effort than new requirements added early in the life cycle because of the volume of change of the project documents and the increasing number of revisions in project documents [166]. Thus, if software requirements change frequently (i.e., high requirements volatility), then they may create significant project uncertainty [161], [166]. One limitation of this approach is that it applies only for specific cases because the measurement focuses on specific types of requirements changes [166]. Hence, systems engineers need to study the effectiveness of categorizing types of requirements changes and their benefits [166].

Other measurements related to requirements volatility in the literature include volatility in scope [167], technical requirements and design changes because of regulatory requirements, schedule changes due to poor or missed requirements, and defects due to inadequate requirements [168]. In regard to the requirements volatility measurements, a review of the literature shows that the common practice in the software industry is to keep the rate of changes at less than 1 for reducing the risk of requirements uncertainty and requirements creep [169], [170]. We will use this value for comparison in the engineering projects.

c. Temporal complexity

In Section 3, we briefly discussed Remington and Pollack's view on temporal complexities. They defined temporal complexities as external complexities associated with market volatility, funding mechanisms, political uncertainty, and financial regulations [84]. In other words, temporal complexity reflects the level of obscurity related to the volatility or disruption caused by the environment (market conditions, social norms, political pressure, regulatory requirements, and technical issues) during the course of the project [84]. Lee and Xia also believe that environmental conditions contribute to project

complexity [85]. Consequently, temporal complexity could influence directional complexity and impact project outcomes.

Over the course of a software development life cycle, software usually has changes to LOC because of changes to requirements, bug fixes, security patches, and optimization of code. Code churn is a measure that quantifies this change [171]. Nagappan and Ball (2005) show that the amount of churn and the temporal extent of churn are indicators of software defect [171]. In other words, they conclude that "code that changes often in before the release will likely have more defects in after the release than code that changes infrequent over the same period" [171]. One drawback of this approach is that it is case study-specific for a large size of software (approximately 44 million LOC [172]).

d. Technological, Social, Cultural, and Task Complexity

Brockmann and Girmscheid (2007) propose three complexity measurements related to engineering projects: 1) social complexity, 2) cultural complexity, and 3) task complexity. Social complexity is determined by the number and diversity of stakeholders collaborating in the project [156]. Cultural complexity depends on the history, experience, and decision-making processes of the team members [156]. Task complexity is determined by the number of activities in terms of time and space required for the coordination and execution of a task [156]. According to Brockmann and Girmscheid [156], elements of task complexities include "structural complexity, technical complexity, directional complexity, and temporal complexity." We have discussed structural complexity and temporal complexity in the previous sections.

As stated in the structural complexity section, Williams [83] believes that uncertainty in goals and methods contributes to project complexity due to users' requirements being difficult to specify and not frozen in the initial prototypes.

Remington and Pollack (2008) define directional complexity as the level of perplexity related to the incoherence of project goals and methodologies among stakeholders [84]. For example, when the project goals and scope are not understood or agreed upon among stakeholders, the project becomes very difficult to manage due to the

uncertainty in project goals, project methodologies, and the lack of urgency and inflexibility among project team members [84].

Heydebrand [172] describes technological complexity and skill structure of an organization in terms of division of labor according to organizational roles. Organizational roles represent different types of job positions in an organization.

Remington et al. (2009) list a number of factors that appear to contribute to project complexity [16]. These factors include the following: goals, stakeholders, management processes, work practices, interfaces and interdependencies, technologies, and time [16], [57]. A summary of these factors is shown in Table 12.

Table 12. Factors influencing project complexity. Adapted from [16], [57].

Factor	Explanation
Goals	Undefined or inadequate project goals both at a strategic level and at an operational level contribute to project complexity.
Stakeholders	The number of stakeholders and how they communicate the information among them affect project complexity.
Management process	Management decision-making processes, relationships between management, stakeholders, team members, and suppliers; and intersecting of activities and methods influence project complexity.
Work practices	The project's organizational structure in terms of division of labor, appointment of personnel, and the level of stress on the personnel to meet project objectives adds project complexity.
Interfaces and interdependencies	The number of systems and subsystems that are integrated in the project, the different assumptions across these systems, the multi-organizational and schedule interdependencies between activities, the work of upgrading and rebuilding, and the size and meshes of the project all contribute to project complexity.

Factor	Explanation
Technology	The need to accomplish tasks by a variety of technologies and the interdependencies between teams, between a network of tasks, and between different technologies contribute to project complexity.
Time	Projects with compressed schedules or unusually long durations and projects requiring multiple phases affect project complexity.

Table 13 summaries the works of some researchers related to the four different approaches to measure project complexity.

Table 13. Summary of researchers' works related to the four different approaches to measure project complexity

Year	Author	Source of Project Complexity
1996	Baccarini [14]	Organizational complexity and technological complexity
2009	Schwandt [158]	Organizational complexity
2010	Wood and Ashton [157]	Organizational complexity
1999	Stark et al. [155]	Requirements volatility
1999	Malaiya and Denton [159]	Requirements volatility and defect density
2000	Pfahl and Lebsanft [161]	Requirements volatility
2001	Feland and Leifer [170]	Requirements volatility
2005	Parsons-Hann and Liu [162]	Requirements complexity
2010	Azim [163]	Geographical distribution of project teams

Year	Author	Source of Project Complexity
2014	Waber et al. [164]	Geographical distribution of project teams
2006	Nurmuliania, Zowghi, and Fowell [166]	Requirements volatility
2010	Peña and Valerdi [168]	Requirements volatility
2002	Lee and Xia [85]	Risk on project environmental conditions
2005	Nagappan and Ball [171]	Code churn
2007	Brockmann and Girmscheid [156]	Social complexity, cultural complexity, and task complexity
1999	Williams [83]	Structural complexity and uncertainty in goals and methods
2008	Remington and Pollack [84]	Directional complexity and temporal complexity
1973	Heydebrand [172]	Technological complexity in terms of division of labor according to organizational roles
2009	Remington, Zolin, and Turner [16]	Factors contributing to project complexity (goals, stakeholders, management processes, work practices, interfaces and interdependencies, technologies, and time)

In sum, previous literature indicates that many researchers from different perspectives have attempted to develop approaches that identify project complexity and its impact on the project. Clearly, the concept of project complexity is broad. There are no widely agreed upon complexity measurement standards for complex projects. As described in this chapter, complexity metrics for software and system development projects have

many variations. However, some of these concepts and methodologies could help develop a practical way to measure complexity. For example, measure of requirements volatility, defect density, and the measure of geographical distribution of project teams are practical measures of project complexity. The approach of this research includes a model that calculates the degree of complexity using the measurement concepts from the four different approaches to measure project complexity, the four kinds of complexity, and the seven building blocks of complexity, which are described in this chapter.

8. Review of Additional Literature Related to Engineered Systems

Some additional literature is relevant to this dissertation. Simon (1962) proposes that the degree of hierarchy is one way to characterize the complexity of a system [100]. Degree of hierarchy refers to the number of layers in which the system can be decomposed [100]. Broadly speaking, Simon [100] observes that the more layers in the system, the more complex the system becomes.

In the shipbuilding industry, a ship's design must satisfy criteria related to performance, cost, safety, environmental regulations, passenger comfort, and customer requirements. Meeting all of these requirements call for complex designs and a complex design process. Caprace and Rigo (2010) propose a complexity metric for practical ship design [173]. The goal is to provide ship designers with information related to design complexity throughout the design process so that they can achieve efficient design during the first prototype. This complexity metric includes a combination of the shape complexity, the assembly complexity, and the material complexity [173].

Complexity in information technology (IT) systems can reduce efficiency and quality in production, reducing a company's flexibility to adopt new technology. Leukert (2011) proposes that the level of complexity in IT systems depends on the function, interface, data, and technology of the system [174].

Some of the recent papers published by several scholars about complexity measurements include "architectural complexity of Systems-of-Systems," "cyclomatic complexity," and structural complexity. Domerçant et al. (2011) propose a model to measure "architectural complexity of Systems-of-Systems" by "the number of functions,"

"the number of network interfaces used to transmit data," and "cyclomatic complexity" [136], [175]. Malone and Wolfarth (2013) propose a model that measure "cyclomatic complexity" and use this measurement to estimate the development cost of a software project [176]. Tie and Bolluijt (2014) propose a framework to measure project complexity that includes "11 contextual factors and 10 inherent project characteristics" [177]. Schuh et al. (2016) propose a conceptual framework that integrates "a set of complexity drivers and a method" to evaluate complexity based on these drivers [178]. Fang et al. (2017) develop a network risk model to measure project risks and their interactions [179]. Ellinas et al. (2018) propose using the structure of the activity network diagrams to measure the structural complexity of projects [180].

Clark and Jacques (2012) propose that a possible way to determine complexity in dynamic systems is to observe changes in kinetic energy [35]. This measurement allows system engineers to understand system risks [35].

Tamaskar et al. (2014) have developed a method to accurately predict development costs and schedules for aerospace products [181]. As industry experts, Tamaskar et al. [181] have proposed a framework to measure system complexity that is based on the size of a system and interactions between system components.

Based on the papers published by several researchers in recent years, the theme of complexity measurement still focuses on models that consist of complexity drivers and indicators in various types of complexity (e.g., structural complexity, inherent complexity, software complexity, technological complexity, and organizational complexity). Many complexity measurement frameworks and models are very complicated and require project managers and systems engineers to invest an extensive amount of time and resources to gather and analyze the data before they can start to use the model to measure complexity. A summary of the work of several scholars related to measures of complexity is shown in Table 14.

Table 14. Complexity measurements in engineered systems.

Year	Authors	System Complexity Measurement
1962	Simon [100]	Measure a system's complexity by the level of hierarchy.
2010	Caprace and Rigo [173]	Measure complexity in ship design based on the shape complexity, assembly complexity, and material complexity.
2011	Leukert [174]	Measure IT systems complexities based on four factors: function, interface, data, and technology.
2011	Domerçant et al. [175]	Measure "architectural complexity of Systems-of-Systems" by a combination of "the number of functions," "the number of network interfaces used to transmit data," and "cyclomatic complexity."
2012	Clark and Jacques [35]	Measure the level of complexity in dynamic systems by observing changes in kinetic energy.
2013	Malone and Wolfarth [176]	Measure "cyclomatic complexity" and use this measurement to estimate the development cost of the project.
2014	Tie and Bolluijt [177]	Measure project complexity by proposing a framework that includes "11 contextual factors and 10 inherent project characteristics."
2014	Tamaskar et al. [181]	Measure the degree of complexity in engineered systems based on size of the system and interactions between system components
2016	Schuh et al. [178]	Measure project complexity by proposing a conceptual framework that integrates "a set of complexity drivers and a method" to evaluate complexity based on these drivers.
2017	Fang et al. [179]	Measure project risks and their interactions using a network risk model.
2018	Ellinas et al. [180]	Measure the structural complexity of projects using the structure of the activity network diagrams.

Moreover, systems engineers and program managers are interested in how they can apply complexity measurement at the mission level [182]. We briefly mentioned in Chapter I that mission engineering and SoS engineering are two areas in which the U.S. Navy must confront complex systems. In mission engineering, the focus is on mission success parameters that drive requirements [182]. In general, a mission has a set of operational activities to achieve a goal [182]. Mission engineering defines the operational tasks or activities and then assigns people, systems, or organizations for executing those tasks [182]. For instance, we can imagine a mission to seize an airport. Mission engineering may consider a solution to this problem that includes ground-based vehicles, drone swarms, robots, and soldiers [182]. From the mission engineering perspective, these individual systems (ground-based vehicles, drones, robots, and soldiers) are components of the overall system, which serves the end-to-end mission of airfield seizure [182]. Consequently, mission engineering is using SoS methodology, which is complex [182]. Another example of mission engineering is the Naval Integrated Fire Control Counter Air (NIFC-CA) project [183]. This project is using an SoS engineering approach to extend the Navy battlespace to the maximum kinematic range of the weapons (i.e., focusing on over-the-horizon targets) [183].

9. Extension of Previous Work

Complexity is "a characteristic of a technical system being developed," and is usually "created by the interaction of people, organizations," and the environments that are "part of the complex system surrounding the technical system" [58]. Many studies have covered a number of complexity measurements in systems, software, and engineering projects that are based on size, diversity, interdependence, dynamism, and adaptivity. However, many of the proposed complexity measures merely serve as starting points for "systems engineers seeking complexity modeling approaches to SE" [81]. With the increasing focus on the value and cost of the system across the acquisition life cycle, models for complexity analysis, analysis of system affordability, and techniques for reducing system complexity are becoming more important in the DOD acquisition arena. There exists a gap between what is available in the DOD acquisition programs today and the complexity measurement models that could be used in early phases of developing

military acquisition products [129]–[131]. For example, according to the U.S. Government Accountability Office (GAO) [130], many DoD programs are underestimating the difficulty of software development efforts. A lack of knowledge of understanding the system requirements and the failure to collect and analyze metrics necessary in the early phases of development to measure progress can increase the risk of budget overruns and schedule slippage due to system development difficulty.

Furthermore, stronger management practices such as using metrics to oversee development progress are needed to reduce the risk of requirements creep, which in turn reduces the complexity of the design processes. Hence, the application of complexity metrics can improve DOD's software-intensive weapon acquisitions [129]. When dealing with complexity and uncertainty in the early phase of a project, the project manager needs metrics that focus on the schedule, the cost, the number of changes in requirements, the number of hours of testing, and the number of unresolved defects of the project [129]. The project manager can use these metrics to create both the project profile and the complexity profile. From these profiles, the project manager can select project cycles, methodologies, and project management strategies. Without such discipline, software acquisition programs may face difficulty in meeting cost and schedule targets [129]. Thus, there is a need to have complexity metrics that focus on the relationship between project complexity and project performance outcomes [135].

Moreover, in recent years, DOD has focused on system affordability for weapon acquisition programs. A review of the relevant literature reveals that there is a gap in the current SE body of knowledge concerning the topics of system affordability and SE advanced measurements [184]. In the topics of system affordability and advancements of SE measurements, there is a need of a use-case study to link between complexity to the cost and schedule of complex systems engineering projects. To study the relationship of this link, we need to identify a model of complexity and measurement methods to analyze project complexity and to provide insight for improvement of system affordability.

Therefore, our effort in this dissertation is to narrow the gap in the body of knowledge of complexity and SE by proposing a new complexity measurement model derived from literature that relates to the topics of system affordability and SE advanced

measurements. The new complexity measurement model is derived from the four major sources of project complexity (structural complexity, temporal complexity, organizational complexity, and technological complexity). This new model results in creating a complexity profile for project managers and systems engineers to pinpoint and then to study what aspects of the engineering project are potential risks of cost overrun and schedule slippage. As a result of this study, systems engineers and project managers can take actions to reduce or mitigate complexity of the SoI, and thus, improve system affordability.

D. CHAPTER SUMMARY

Chapter II surveyed the history of complexity science and highlighted major works about measures of complexity from the scientific literature. Chapter II showed that complexity results from the numerosity, connectivity, interactivity, diversity, and adaptivity of a system as well as from its environment [18], [28], [29], [116]. In addition, Chapter II presented some of the flaws and drawbacks of existing measures of complexity as well as several ideas to address these gaps. It discussed complexity as applied in software applications and system development projects. Through a review of the literature, we showed that complexity can influence projects in terms of difficulty of understanding, verification of work performed by the project team, and control of project resources [166]. In fact, complexity contributes to project failure [165]. Understanding the different types of complexity can help project managers make better decisions about the ways to manage complex projects. Finally, Chapter II discussed some important properties of measures relating to the seven building blocks of complexity.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROJECT COMPLEXITY MEASUREMENT MODEL (PCMM)

> The quest of any model is to ease thinking while retaining some ability to illuminate reality.
>
> —John H. Miller and Scott E. Page [46],
> "Complex Adaptive Systems:
> An Introduction to Computational Models of Social Life"

As noted in Chapter II's literature review, complexity can be increased by conflicting requirements and the mingling of problem elements, multiple constraints, diverse stakeholders, and demanding stakeholders during the engineering process. Complexity also can be increased by the interactions between decisions and the analytical tasks during the system development processes. This realization strengthens the validity of creating a complexity profile of an engineering project that consists of multiple complexity measures for project risk assessment. This chapter describes a practical approach to measure complexity. The approach includes identifying a set of complexity metrics drawn from the literature review. Next, we develop the project complexity measurement model (PCMM) and associated methods to measure complexity. The PCMM presented in this chapter is derived from literature related to the sources of complexity and the four major types of project complexity (structural complexity, temporal complexity, organizational complexity, and technological complexity) presented in Chapter II. Furthermore, the literature on complexity and project complexity in Chapter II provides evidence of the validity of the PCMM and the research methodology presented in this chapter. This chapter includes the formulation of the system and project complexity metrics. This chapter also discuss system and project properties as well as analysis methods used throughout the dissertation. The analysis methods depend on a project profile, other known program parameters, and historical data. Finally, we discuss how program managers can use the complexity measures presented in the model to take actions to reduce complexity. A roadmap of this research approach appears in Figure 6. We will discuss each of these steps in the next section of this chapter.

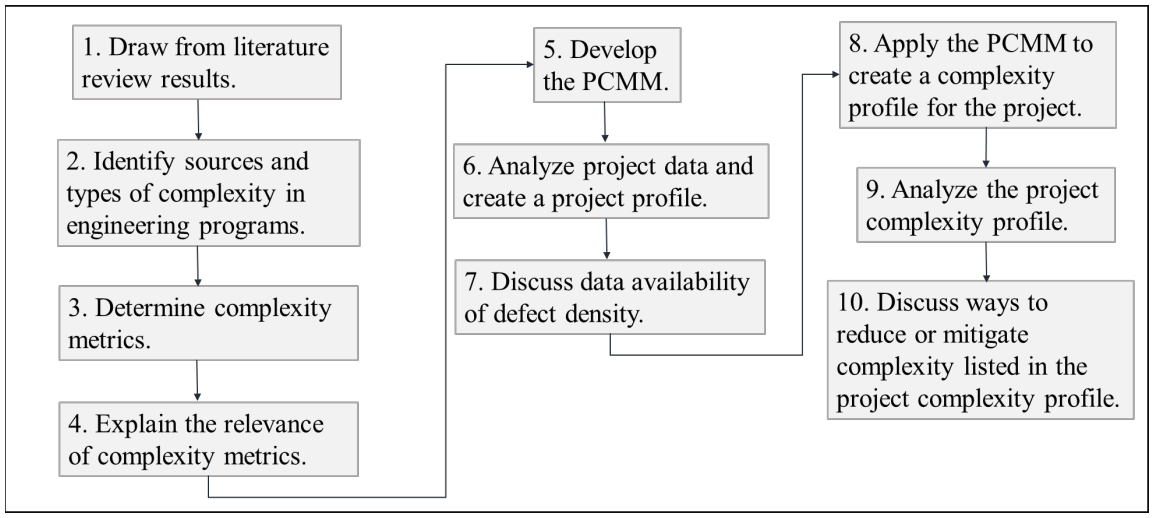

Figure 6. Research approach roadmap

A. RESEARCH APPROACH

The outcome of the PCMM is a complexity profile for the SoI. In Figure 7, we provide the overall processes to create the complexity profile of a SoI.

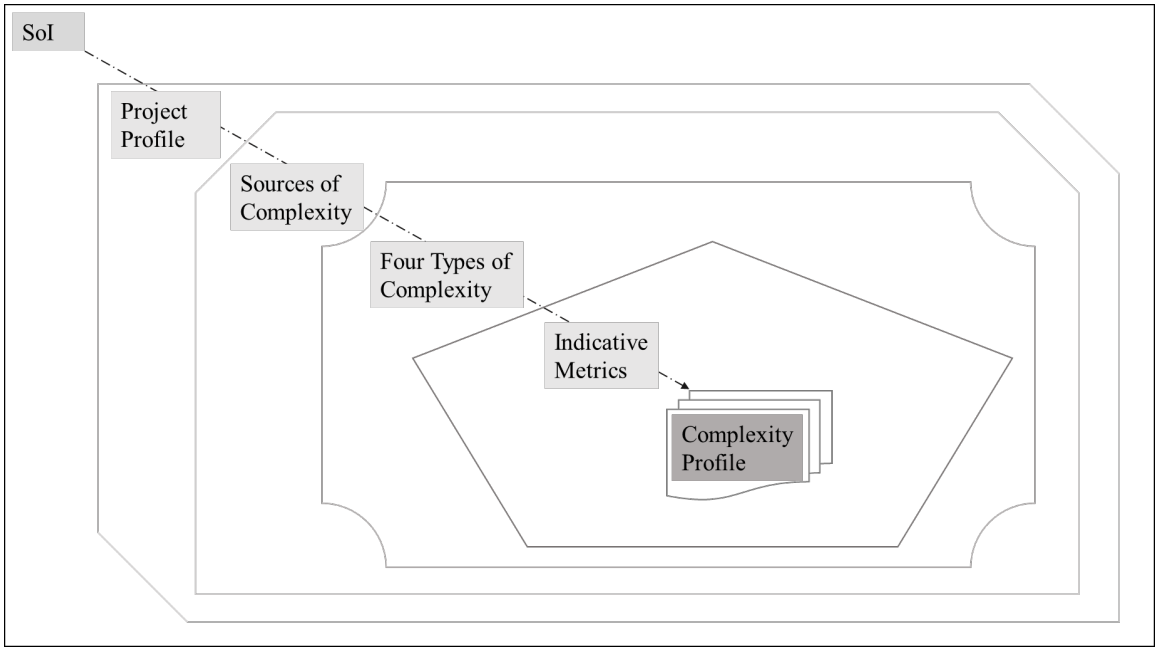

Figure 7. Overall processes to develop the complexity profile for the SoI

In step 1 of Figure 6, we draw results from the literature review from Chapter II. Four types of complexity were drawn from four academic publications (Baccarini [14], Williams [83], Brockmann and Girmscheid [156], and Remington and Pollack [84]) as follows: 1) structural complexity, 2) organizational complexity, 3) temporal complexity, and 4) technological complexity.

In step 2 of Figure 6, we identify sources and types of complexity in an engineering program. We explained in Chapter II that system architecture, software architecture, and software size are project elements that can contribute to structural complexity in engineering projects. In a system development process, organizational complexity is increased by the number and diversity of stakeholders, size and structure of the project team, nature of the procurement process, and information flow between activities in terms of information exchanges between organizational units. Furthermore, we explained that we can use a project management framework such as the Project Management Body of Knowledge (PMBOK) [185] for classifying the sources of complexity and subsequently relating them to the types of complexity and complexity metrics to study project complexity.

The PCMM, which consists of a set of complexity measures, creates a complexity profile of the SoI that is useful for the assessment of project risk and program complexity. The PCMM and associated methods to measure complexity can help systems engineers and program managers reason through their decisions related to systems engineering. Similarly, the PCMM and associated methods to measure complexity are designed to shed more light on Williams' [83] approach for measuring the complexity of the underlying structure of the project. The PCMM measures the structural complexity of a project in two different ways: 1) the number of personnel required for the project to develop the SoI, and 2) the number of defects per project size.

In addition, the PCMM extends the work of both Baccarini [14] and Lloyd [90] for measuring the complexity of organizational structure in a complex project. The PCMM measures the organizational complexity of a project in two ways: 1) The patterns of communication in terms of their frequencies and the level of importance of the communications among organizations, and 2) The geographical distribution of teams.

Moreover, the PCMM also follows the approaches of Remington and Pollack [80] and Stark et al. [155] for measuring the temporal complexity in terms of requirements volatility during the engineering project. Finally, the PCMM uses the approaches of both Baccarini [14] and Lloyd [90] for the assessment of technological complexity by presenting a metric that measures the type of skilled staff needed for the development of the SoI. In sum, the PCMM and associated methods to measure complexity make contributions in at least two major ways: 1) a new quantitative approach to assess project complexity, and 2) advancement in the state of practice in the project risk assessment of Navy software systems.

In step 3 of Figure 6, we develop the indicative metrics of the four types of complexity of the SoI that are important to the IPT, program managers, and systems engineers. Each type of complexity is associated with several complexity factors that contribute to the complexity of the system development process [16], [57], [83], [85], [86], [156], [162], [164], [173]. Chapter II revealed that complexity is influenced by many factors. The list of complexity metrics is long, and in the end not very practical to use because it would be difficult to measure all complexity metrics due to budget constraints and schedule constraints. Moreover, it would be impossible to agree on the exact scope of each type of complexity or on the definition of complexity as numerous authors proposed different taxonomies, characteristics, and attributes of complexity. Current literature provides some evidence on the validity of the above arguments. Therefore, we have chosen to focus on the major sources of complexity from the four types of complexity in the SoI.

Project managers and systems engineers are concerned about deliverables and requirements of the SoI. They need to keep the project within budget and maintain the schedules by minimizing project risk. To identify project risk, systems engineers and project managers select certain complexity metrics and measure them. For instance, the project manager faces great challenges in managing a project in a virtual environment. To improve the likelihood of success and predictable project outcomes, the project manager needs to understand the effects of the fact that the virtual team is located in multiple locations in multiple time zones on the project schedule and the quality of the deliverables.

For an assessment of the risk of the structural complexity, we choose complexity metrics such as size of scope, project duration, and number of requirements related to product quality control. These complexity metrics influence the number of personnel required for the development of the SoI.

In organizational complexity, we select size of the project team, number of locations of the project team, number of tasks, team communication activities, and experience of the participants as the main complexity metrics for project risk. These complexity metrics are measured by the geographical distribution of teams and by the patterns of communication in terms of their frequencies and the level of importance of the communications among organizations.

In temporal complexity, we pick project stability as the foremost metric for project risk. Requirements volatility measures project stability in terms of the number of requirement changes as a function of the duration of the development phase of a project.

In technological complexity, we select the integration of technical processes and dependencies between tasks as the leading complexity metrics for project risk. The measure of the number of different job position types provides some insights about the complexity of integrating technical processes and executing interdependencies tasks in the development phase of a SoI.

In Table 15, we present the indicative metrics of the four types of complexity of the SoI.

Table 15. Indicative metrics of the four types of complexity of the SoI

Type of Complexity	Indicative Metrics
Structural complexity	• Measure of the number of personnel required for the project (e.g., COSYSMO [24]) • Defect density measure
Organizational complexity	• Measure of the patterns of communication in terms of their frequencies and the level of importance of the communications among organizations • Geographical distribution of teams measure
Temporal complexity	• Requirements volatility measure
Technological complexity	• Measure of the number of different job position types required on the project

In step 4 of Figure 6, we need to explain the relevance of the complexity metrics. Let's describe each measure of complexity in order.

1. Structural Complexity

In managing a project's scope, we need to define, verify, and control project work. As noted in Chapter II, COSYSMO measures the person-month required for performing the system engineering tasks in large-scale systems [24]. It is related to the structural complexity of the project because it involves size and cost drivers for systems engineering [24]. We use COSYSMO to measure structural complexity in terms of the functional size of a system, which is expressed in terms of the number of person-month. Drivers of project size consist of the number of requirements, interfaces, algorithms, and use cases [24]. Drivers of project cost include requirements understanding and architecture understanding [24]. These cost drivers are indicative of either uncertainty or an emerging understanding of needs during the engineering process. As noted in Chapter I, uncertainty is one of the characteristics of complexity.

In a software development project, defect density measures the number of defects per KLOC. We categorize defect density as structural complexity because KLOC is a function of a system's functional size. Defect density indicates product quality, project progress, and productivity.

In managing the quality of a software project, we set quality objectives for the project. Quality objectives set during the planning phase can introduce complexities to projects. For example, a project manager requires the software lead to set up baseline measurements of the code walk-through process. A project manager also requires the test lead to provide a monthly report of defect density so the program manager can track the overall quality trend of the products. The quality objectives for the project can possibly cause frustration to the developers, and collecting the baseline metrics can complicate and lengthen the duration of the code walk-through process.

2. Organizational Complexity

In an engineering project, the interchange of information between organizations can be measured by the patterns of communication between organizations in terms of the frequency and the level of importance of their interactions. It is related to the organizational complexity of the project because it indicates the information flow between tasks and activities. It shows the connectedness and interdependence among organizational units during the system engineering processes. For example, in managing a project's procurement and meeting deadlines, project managers are concerned with the timely completion of the project and must handle the process of acquiring third parties' products and services. The number of interdependent organizations involved in meeting a project's deadlines and in procurement execution can influence the complexity of the project. Literature seems to confirm that the interdependent relationships between organizations contribute to project complexity [28], [93], [94]. Hence, project managers find that the patterns of communications in terms of the frequencies and the level of importance of the communications among organizations are crucial for the success of each program.

The geographical distribution of teams measures the number of locations, time zones, and sites of all organizational units involved in the engineering project. It is related

to the organizational complexity because it measures the number of sites involved in the engineering development process, which is related to the team size in a virtual organization. In general, when managing a project in a virtual organization, coordinating, acquiring requirements, and executing project tasks with project teams at various sites across multiple time zones are all very difficult [162], [186]. In fact, distributed teams need more coordination than physically co-located teams because of the multiple sites and the barriers associated with adopting technology across multiple sites. The Allen curve supports the idea that a strong negative correlation exists between physical distance and the frequency of communication between work stations [119], [187]. For example, in managing a project's human resource, project managers must handle all necessary processes for organizing, managing, and leading the project team. Factors such as team size and geographical distribution of project teams can contribute greatly to project complexity.

3. Temporal Complexity

Requirements volatility measures the number of changes in requirements (new requirements, modifications of requirements, and deletion of requirements) during a specific period of time in the development phase of a project. It is related to the temporal complexity because it measures project stability in terms of the number of changes in requirements as a function of the duration in the development phase of a project. High volatility of changes in requirements reflects uncertainty of the design of the system's architecture and poor understanding of the environment and system. If the requirements are changing too quickly, this introduces problems in all areas of the system development, especially in meeting a deadline and keeping within budget. For instance, during the development and integration of a system, we often have change requests to add, modify, or delete requirements. These change requests reflect changes in customer's priorities, changes in the budget of the project, and changes in developers' understanding of the requirements and product [84]. Requirements volatility can cause product defects due to changes in either hardware or software or both in a system, and reflects decision-making under uncertainty.

4. Technological Complexity

One way to measure technological complexity in a project is by counting the number of different job position types because of the dependencies between tasks and the diversity of job skills required during the integration of technical processes in the development phase of a SoI. For example, the number of different job position types required on a project is likely to foster differences in job assignments and outcomes, and, thus, increased complexity.

To measure the degree of influence of each of the aforementioned four types of complexity and their indicative metrics, in the next section we introduce the PCMM and associated methods to measure complexity. This is indicated in step 5 of Figure 6.

B. METHODOLOGY

In this section, we present the PCMM to assess the degree of complexity based on the indicative metrics of the SoI shown in Table 15. We also apply the PCMM to measure complexity of a Navy software acquisition program.

In general, traditional project management practices encourage linear thinking and plenty of structure and control. Complex projects are different from traditional projects in a variety of ways, including:

- Project size,
- Project cost,
- Scope,
- Deliverables,
- Uncertain requirements,
- Multiple locations across multiple time zones,
- Large virtual teams, and
- Numerous complex interactions.

Complex projects are usually large in scope, and the initial statement of work is often only partially complete. For example, research and development (R&D) projects are complex because project teams are not 100 percent sure where the project is leading toward

something meaningful, and they do not know what the project will cost nor if and when they will achieve something useful in the project. As a result, large, complex projects often have large cost overruns and schedule slippages. For instance, the Denver International Airport was a complex, huge construction project, and it had significant construction cost overruns (about $58.4 million higher than the original budget) and schedule slippages [188]. To manage risk of schedule slippage and budget overruns in meeting the necessary project deliverables, program managers need to measure project complexity.

1. Project Complexity Measurement Model (PCMM)

The PCMM consists of six complexity metrics derived from four major types of project complexity discussed in previous section.

a. Measure of Number of Personnel Required for the Development Effort

A rule of thumb of cost in engineered systems is to allocate 15% of total program effort to systems engineering [189], [190]. One way to estimate the number of personnel required for the development effort is to use the COSYSMO [24] for an estimate of how long and how many people it takes to complete the project. The amount of person-month required for a project (PM) is estimated as follows [24]:

$$PM = (A)(Size)^E \prod_{i=1}^{n}(EM_i) \tag{3.1}$$

where *PM* denotes effort "in person-month" for a formal schedule [24], "*A* denotes a calibration constant derived from historical project data, *size* denotes the summation of the weighted four size drivers (requirements, interfaces, algorithms, use cases)" [24], *E* denotes the efficiency factor, "*n* denotes the number of cost drivers, and EM_i denotes effort factor for the i^{th} cost driver (i.e., requirements understanding, technical risk, and process capability)" [24].

b. Defect Density Measure

The defect density measure (*DDM*) is an indicator of product quality, progress, and productivity. It is defined as follows:

$$\text{DDM} = (ND_i) \,/\, (\text{P}S_i) \tag{3.2}$$

where ND_i denotes the weighted average number of defects during the i^{th} period, i is measured by months, and PS_i denotes the project size at the i^{th} period, which is measured by KLOC for software projects.

c. Organizational Complexity Measure

The organizational complexity measure is defined as an index function (*IF*) [157],

$$IF = \left(\sum_{ij=1}^{n}(w_{ij})(f_{ij})/(5F)\right) \tag{3.3}$$

where $w_{i,j}$ denotes a factor of importance of the communications between two nodes *i* and j [157], f_{ij} denotes the frequency of the communications between two nodes *i* and *j, F* represents the total number of communications between two nodes *i* and *j,* and *n* denotes the number of organizations involved in the communications [157]. Note that the term of *5F* is a normalized factor of both frequency and importance of the communications. The range of *IF* is between 0 and 1. While a zero value of *IF* has no contribution to the complexity, a value of one in *IF* means a maximum contribution to the organizational complexity [157].

We define frequency as the number of times that an organizational unit sends information to another organizational unit during the period of the development cycle, which is measured by weeks. We define level of importance based on how important the information flow is to the organizational unit. We use a Likert scale [32] of 1 to 5 to map the level of importance in information flow as shown in Figure 8.

Level of Importance	Interpretation
1	Not important (i.e., receiving units use this information as a status only)
2	Slightly important (i.e., receiving units use this information as heads up to plan some tasks in the program)
3	Moderately important (i.e., receiving units use this information to update project metrics)
4	Important (i.e., receiving units needs this information to take actions on some tasks)
5	Very important (i.e., receiving units needs this information to make a decision on a program)

Figure 8. Likert scale. Source: [32].

d. Geographical Distribution of Teams Measure

The geographical distribution of teams measure (*GD*) is defined as follows:

$$\text{GD} = (l + t + s) \tag{3.4}$$

where *l* denotes the total number of locations of all organizational units, *t* denotes the total number of time zones of all organizational units, and *s* denotes the total number of sites of all organizational units.

e. Requirements Volatility Measure

The requirements volatility measure (*RVM*) is defined as follows:

$$\text{RVM} = (\text{CR}_i) / (\text{BR}) \tag{3.5}$$

where CR_i denotes the number of changes in requirements (new requirements, modifications of requirements, and deletions of requirements) during the i^{th} period, *i* denotes each release in months, and *BR* denotes the baseline of the software release, which is measured by the number of requirements.

f. Number of Different Job Position Types

The number of different job position types (*JPT*) is defined as follows:

$$\text{JPT} = (\sum_{i=1}^{n} p_i) \tag{3.6}$$

where p_i denotes the number of job position type identified in the i^{th} organizational unit and *n* denotes the total number of organizational units in the project.

2. Acquisition of Test Data to Perform the Complexity Measurements

Not all of the PCMM metrics can be measured at the beginning of the project. In fact, two of the six measures of the PCMM depend on having test data to compute the values of complexity. To resolve this issue, project managers can use the project plan of the previous project or a project before with a similar size and make certain assumptions on the current project regarding the levels of difficulty in the implementation of system requirements, interfaces, algorithms, and use cases. With these assumptions, the project manager can compute the complexity values of four PCMM measures: (1) the number of personnel required for the development effort, (2) organizational complexity, (3) geographical distribution of teams, and (4) the number of different types of job positions.

The PCMM measure of defect density depends on having test data. This issue of not having test data to measure defect density can be mitigated by relying on the experiences of the subject-matter experts on the history of the project and leveraging project data from the previous project or from a project before with a similar size for insight of the current project. The project manager may need the testers to run some simulations of the system to obtain test data. In some cases, the project manager can also obtain the test data after the testers completed their initial tests during the development phase, which can occur in a few months after the project started.

The project manager can estimate the overall defect density of the project in the beginning of the project based on the previous project data or data of projects of a similar size and allocate at least 10% of project reserves to cover the residual risks in the project. Furthermore, the project manager can monitor the number of changes request and implement configuration management control.

The PCMM measure of requirements volatility also depends on having data after the project has gone through a period of development. Similarly, this issue of not having data beforehand can be mitigated by leveraging the requirement traceability reports from the previous project or from a project before with a similar size for insight in the current project. The project manager can schedule peer reviews on the software requirement

specifications (SRS) earlier in the development phase and establish an initial requirements traceability report of the project.

In addition, the project manager can estimate the overall requirements volatility of the project based on the previous project data and allocate at least 10% of project reserves to cover the residual risks in the project. Furthermore, the project manager can control the number of requirements change during the development period by the configuration control board.

3. Methodology of applying the PCMM to Assess Project Complexity

We introduce a complexity study method (CSM) to assess project complexity. This method involves five steps as shown in Figure 9.

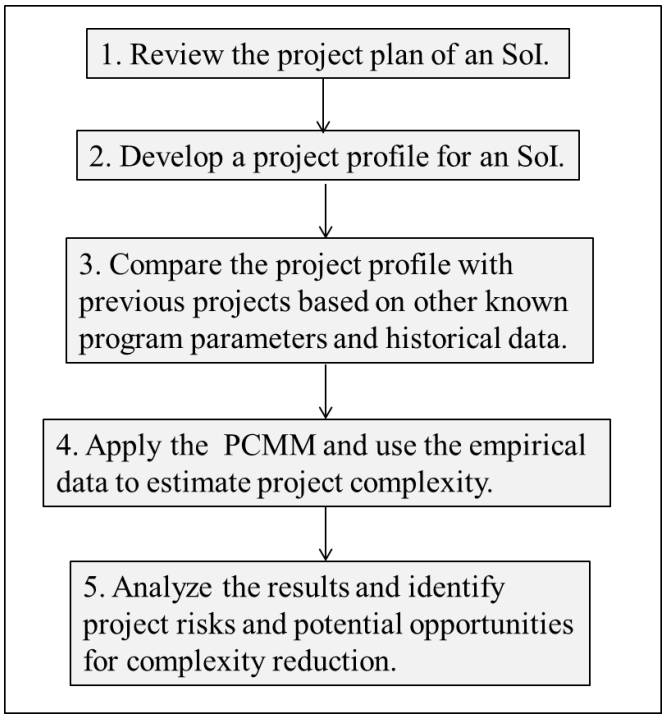

Figure 9. CSM for applying the PCMM to complex projects

In step 1 of the CSM, we review the SoI project data from the project plan and develop the project profile. The work of reviewing the project data requires studying and

identifying key aspects of project attributes and characteristics. For example, program managers strategically seek to determine the following:

- Whether or not any organizations have developed similar systems before,
- Whether system quality is measurable,
- The number of organizations, users, and stakeholders involved,
- The kind of technology used (hardware and software),
- The number of detailed design documents and use cases,
- The number of functional requirements, conflict requirements, and inadequate requirements,
- Whether control is decentralized and distributed across physical systems and cyber space, and
- Known risks and future opportunities.

We follow step 2 of the CSM. The outcome of this step is a project profile as shown in Figure 10 that identifies key elements of system characteristics and important factors that affect project cost, schedule, performance, and risk. In step 3 of the CSM, program managers use this project profile to analyze and to select other known program parameters and historical data for making certain assumptions and decisions related to the complexity profile for the project. Figure 10 presents the contents of the SoI project profile.

Contents of an SoI Project Profile
• Estimated annual budget of the project in dollars • Number of team members on the project • Duration of the development phase • Number of functional requirements at project start. • Number of implemented requirements at the end of the project • Number of change requests approved (new requirements, modifications of requirements, and deletions of requirements) • Number of implemented requirements designated as either "easy," "nominal," or "difficult" • Level of requirements understanding of the project team [24] • Level of technical risk of the SoI • Level of process capability of the project team [24] • Number of external interfaces and level of difficulty of implementing the interfaces • Number of critical algorithms [24] and level of difficulty of developing the algorithms • Number of use cases and level of difficulty of testing the use cases • Software project size in terms of LOC

Figure 10. Contents of an SoI project profile

With a project profile, the program manager performs step 4 of the CSM to assess the degree of complexity of the SoI project. For instance, as part of measuring defect density in a software project, program managers need both the number of defects and LOC for a release. Because program managers do not have the data until they finish building the product, they have to rely on the historical data of similar projects. They have to find a similar project in terms of project size, project duration, and number of functional requirements from an in-house project repository system (e.g., Microsoft SharePoint and pragma Systems processMax). A repository system contains many project artifacts such as test reports, decision memos, review documents, project metrics, use cases, and detail design documents. Program managers obtain the historical data from test metrics derived from a monthly test report. Test metrics often include the date, LOC, and the number of cumulative defects and test hours for each release in each month. As part of managing the quality of a software project, the test lead often collects test data from testers and then generates test metrics monthly according to the test plan. He or she submits test metrics to

program managers for monthly review and trend analysis. The test lead also posts test metrics on an in-house project repository system such as Microsoft SharePoint as part of the project archive process. Hence, with the available data from past programs, program managers can measure defect density quite easily. Figure 11 shows the steps to develop a project complexity profile using the PCMM.

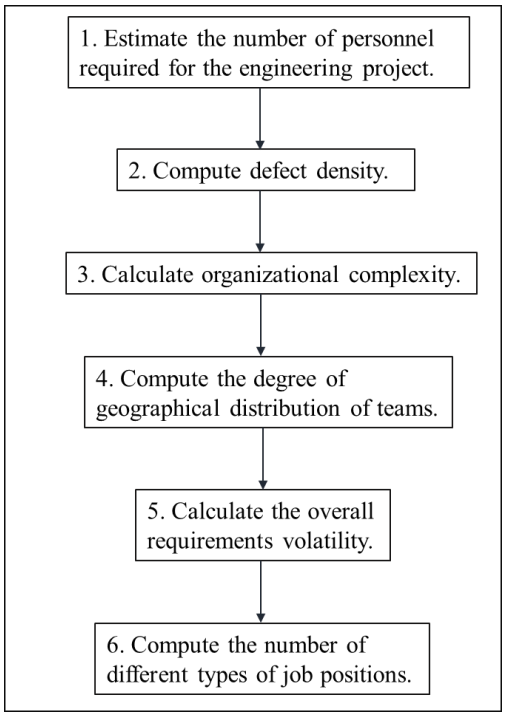

Figure 11. Steps to develop a project complexity profile using the PCMM

Once we develop a project complexity profile using the PCMM, we follow step 5 of the CSM and identify potential opportunities to reduce project risks.

To demonstrate the PCMM, we analyze a Navy software development project. This project uses an evolutionary development method in which the program delivers the software in increments known as "blocks." In this case, we designate this project as the B1 Project. As indicated in step 6 of Figure 6, we analyze the project data and create the project profile. Table 16 shows the B1 Project profile derived from the empirical project data. The project data shown in Table 16 are available when the project starts.

Table 16. B1 Project profile

Project Property	Value	Source of Information
Yearly Budget	$3 million ($4.5 mil/1.5 years)	Project plan
Team Size	20	Project plan
Duration of the development phase	18 months (1.5 years)	Project plan
Baseline Requirement (BR)	1,946 requirements (at project start)	Requirements traceability report (RTR)
Number of requirements at the end of the project	1,721 requirements	RTR
Number of Change Requests (CRs)	528 CRs (151 new, 1 modification, 376 deletions)	RTR
Number of “nominal” requirements	344 requirements (20%)	Applying the 80–20 rule that is based on the interview of program managers of previous programs
Number of “easy” requirements	1,377 requirements (80%)	Applying the 80–20 rule that is based on the interview of program managers of previous programs
Understanding of requirements	high	Project plan
Technical Risk	nominal	Project plan
Process Capability	nominal	Project plan (CMMI level 3 [191])
Number of External Interfaces	2 (all “easy”)	IDD
Number of Critical Algorithms	1 (“nominal”)	Software design document (SDD) and project plan
Number of use cases	32 (all “easy”)	Use case documents and test plan
Number lines of code (LOC)	576,500	Build report (SLOC count)

In the following sections, we follow the steps outlined in Figure 11 and apply them to the B1 Project to demonstrate the method to measure project complexity.

a. Measure of Number of Personnel Required for the Development Effort

As noted in Table 9 of Chapter II, the COSYSMO contains 14 cost drivers. The effort multiplier ratio (EMR) represents the influence on the systems engineering effort. The four most influential cost drivers in the COSYSMO, which have high EMR values, are the following: (1) requirements understanding, (2) architecture understanding, (3) level of service requirements, and (4) technology risk. The least influential cost drivers, which have lower EMR values, are as follows: documentation, number of installations, number of recursive levels in the design, and tools support. Early in the system life cycle, the sources of data for some cost drivers may not be available to the project manager. For software projects such as the B1 Project, the set of cost drivers that significantly drive the systems engineering cost are requirements understanding, technology risk, and process capability, which is shown in Table 17. The data for these three cost drivers are available from the project plan when the project starts.

Table 17. Rating values for cost drivers of systems engineering. Adapted from [24].

	Very Low	Low	Nominal	High	Very High	Extra High	EMR
Requirements understanding	1.87	1.37	1	0.77	0.60		3.12
Technology risk	0.67	0.82	1	1.32	1.75		2.61
Process capability	1.47	1.21	1	0.88	0.77	0.68	2.16

Requirements understanding is defined as the level of comprehension and familiarity of the system requirements by the development team [24]. Based on his or her work experiences, the project manager assesses the level of familiarity of the system

requirements of the development team. The overall degree of understanding of these requirements has an effect on the amount of effort needed for SE.

Technical risk is characterized as the possibility of requiring more SE effort due to immaturity and obsolescence of the technology implemented into the system [24]. This is based on a subjective assessment of the system from the project manager.

Process capability is defined as the ability of the project team to follow defined processes with a certain degree of effectiveness [24]. The Capability Maturity Model Integration (CMMI) [191] is one of many published process models that is used in rating process capability.

The four size drivers that affect the system engineering effort are the number of requirements, number of major interfaces, number of critical algorithms, and number of use cases [24].

To demonstrate the PCMM and associated methods to measure structural complexity, we use the data from the B1 Project profile (Table 16), and the relative weights for size drivers shown in Table 8 of Chapter II to calculate the COSYSMO size [24] for the B1 Project. The results are showed in Table 18.

Table 18. Calculation of COSYSMO size [24] for the B1 Project. Adapted from [24].

COSYSMO size's driver	B1 Project software size
Requirements	(1,377 easy requirements)(0.5) = 688.5 (344 nominal requirements)(1) = 344 Total = 688.5 + 344 = 1,032.5
Interfaces	(2 easy interfaces)(1.7) = 3.4
Algorithms	(1 nominal algorithm)(6.5) = 6.5
Operational Scenarios (use cases)	(32 easy use cases)(9.8) = 313.6
Total COSYSMO size = requirements + interfaces + algorithms + user cases	1,032.5 + 3.4 + 6.5 + 313.6 = 1,356

Similarly, we calculate the effort weight factor from Tables 16 and 17 for the B1 Project as shown in Table 19.

Table 19. The B1 Project's effort weight factor. Adapted from [24].

Effort weight factor	Value
Requirements understanding = high	0.77
Technical risk = nominal	1
Process capability = nominal	1
The B1 Project's effort weight factor = (requirements understanding)(technical risk)(process capability)	(0.77)(1)(1) = 0.77

As described in Chapter II, in a typical software development project, the value of the COSYSMO calibration factors A is 0.325 [24], and the default value of E is 1 [24]. We use Equation (3.1) to calculate the amount of person-month required for the B1 Project and present the results in Table 20.

Table 20. The B1 Project development effort

B1 Project development effort	Value
Effort required to build the system = PM = (calibration factors)(effort factor)(software size)	(0.325)(0.77)(1,356) = 339.34 person-month
B1 Project development effort = (effort required to build the system) / (duration of the development phase)	339.34 / 18 = 18.85 people

Table 20 shows that the B1 Project requires 18.85 people to develop the system in 18 months. The project personnel measure of 18.85 people is very much in line when compared to the project team size of 20 listed in the B1 Project profile (Table 16). As noted

in Chapter II, the average annual salary of a software developer is $145,932. The project plan indicates that the estimated yearly project cost is $3 million. If we divided the annual project cost by the average yearly salary of a software developer, we obtain approximately 20 developers that could be hired per year to work on this project. This result demonstrates that the PCMM and associated methods to measure the number of personnel required for the development effort is analytical and accordant with the measure of structural complexity defined in terms of size and cost drivers for an engineering project.

b. Defect Density Measure

According to COSYSMO [24] and several sources from literature [192], [193], we define project size in terms of the number of LOC, the number of function points, the number of requirements, the number of documentation pages, and the number of test hours. In a software project, high defect density indicates poor product quality, difficult or poor project attributes, and uncertainty of the system [165]. The patterns of requirement changes over time have substantial influence on defect density because software developers must make changes in the code to reflect changes in requirements or to fix bugs in a software module [159], [165]. For example, if the requirement changes occur close to the software release date, this will cause the resulting software to have a higher defect density [159]. Therefore, the product quality will likely go down. High defect density also has a significant impact on the development cost [161]. Fixing software defects after releasing the product is expensive. Project managers and system developers are interested in this measure for project management, risk assessment, and potential assessment of project complexity.

Moreover, knowing the relationship between complexity and defect density is very useful to project managers. Alfadel et al. [194] demonstrate that two software complexity metrics, cyclomatic complexity [136] and Halstead complexity [139], have a linear relationship with defect density metric.

In step 7 of Figure 6, test data to measure defect density may not be available in the early phase of system life cycle due to the evolutionary nature of systems. In that case, test data of previous acquisition programs must be obtained or derived in order to estimate the

defect density of the project. The project manager may need the testers to run some simulations of the system to obtain test data. In some cases, the project manager can also obtain the test data after the testers completed their initial tests during the development phase, which can occur in a few months after the project started.

To illustrate the PCMM and associated methods to measure defect density for the B1 Project, we analyze the test data. In this case, the project has been in the development phase for 18 months. Testers has tested the software during this period. Thus, the test data is available to calculate the defect density for this project. The project plan indicates that the B1 Project is programmed using the C# object-oriented programming language. We obtain the data for this measure for release 0 (baseline) through release 18 (18 months) from a test metric report via Microsoft SharePoint (Table 21). In Table 21, the number of defects for a release (ND_i) is the cumulative number of defects unresolved at the end of the i^{th} release. The defect density (DD) is calculated using Equation (3.2).

Table 21. B1 Project defect density during the 18-month period

Release (*i*)	**Month**	**Cumulative defects unresolved (ND_i)**	**KLOC**	***DD* = (ND_i)/(KLOC)**
0	Oct. 2014	19	512.727	0.037
1	Nov. 2014	33	512.727	0.064
2	Dec. 2014	42	512.782	0.082
3	Jan. 2015	50	512.782	0.097
4	Feb. 2015	55	512.837	0.107
5	Mar. 2015	56	512.862	0.109
6	Apr. 2015	56	512.862	0.109
7	May 2015	56	535.873	0.104
8	Jun. 2015	38	552.103	0.069
9	Jul. 2015	47	552.103	0.085
10	Aug. 2015	67	552.103	0.121
11	Sep. 2015	84	552.213	0.152
12	Oct. 2015	90	575.249	0.156
13	Nov. 2015	103	575.249	0.179
14	Dec. 2015	106	575.397	0.184
15	Jan. 2016	119	575.474	0.207
16	Feb. 2016	126	575.745	0.219
17	Mar. 2016	134	576.0	0.233
18	Apr. 2016	185	576.5	0.32

Figure 12 displays a plot of defect density against the release version number with release 0 being the start of the new project (baseline). From Figure 12, the defect density ($ND_i/KLOC$) is growing fairly steady throughout the development cycle. After 18 months in the development phase (April 2016), the project averaged 0.32 defects per KLOC, which is within the industry standard limit of less than 3 [160]. In general, we expect that an increase in test hours will yield higher defect density during the development phase because more defects are discovered. As shown in Figure 12, the "ND_i//KLOC" increases slightly at the end of the development life cycle because testers have increased their test hours and discovered more bugs before the release of the software. This finding confirms the previous studies [159], [165] showing that the patterns of requirement changes over time have substantial influence on defect density. Hence, the PCMM and associated methods to measure defect density is consistent with the measure of structural complexity defined in terms of size and cost drivers for systems engineering.

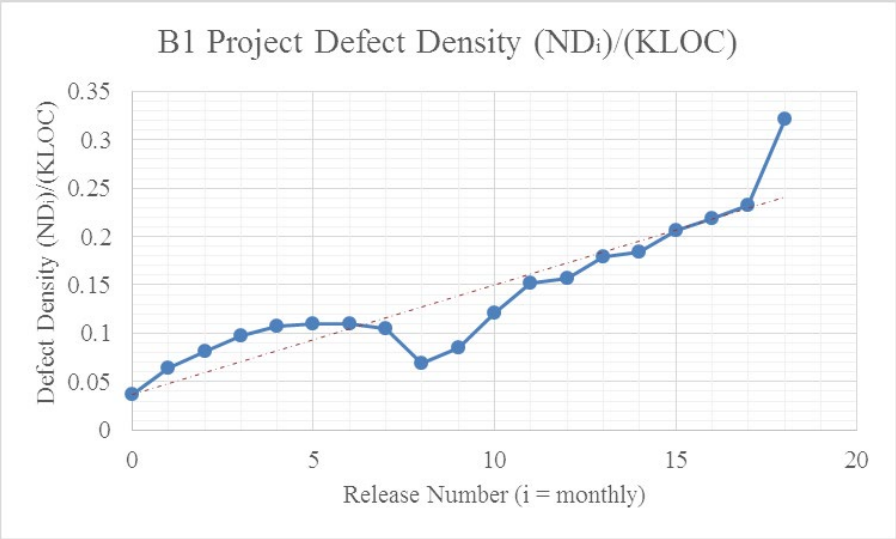

Figure 12. B1 Project defect density metric

c. Organizational Complexity Measure

Broadly speaking, we observe that organizational complexity exists in many DoD acquisition systems. Interactions among multiple organizations and stakeholders in a development team can contribute to organizational complexity. A temporary multi-organizational structure in a program consists of several organizational units that are responsible for funding, managing, and executing program tasks. We define an organizational unit as a distinct group, either internal or external of a company, with a set of roles in the project. These roles are responsible for managing the delivery of work, services, and resources to the project. For example, a contractor unit is an organizational unit that is responsible for delivery of work and services to the project. A resourcing unit (e.g., PMA) is an organizational unit that is responsible for providing funding to the project.

A high organizational complexity measurement generally reflects the following factors [88], [157]:

- Poor channels of communication,
- Poor generation and use of information for decision-making,
- Poorly defined project roles,
- High interdependencies between roles,
- A large number of project stakeholders,
- Poor relationships between the project parties, and
- Difficult relationships with the project sponsor.

A combination of these factors contributes to organizational complexity. In a multi-organizational structure, one way to measure organizational complexity is to analyze the relationships between organizational units. Connectedness and interdependence between organizational units are two primary relationships [101], [102]. Connectedness is the number of units in which one unit directly connects to or interacts with other units. Two connected units may, or may not, interact. They may exchange information via networks. Interdependence is the degree to which one unit's actions control or influence another unit's actions and outcomes, and vice versa [28].

To demonstrate the PCMM to measure organizational complexity in the multi-organizational project, we analyze the B1 Project organizational structure from the project plan. The project plan indicates nine organizational units and lists their responsibilities (Table 22). The data shown in Table 22 are available when the project starts.

Table 22. The B1 Project organizational units and their responsibilities

Organizational Unit	Responsibilities
1. Program Management Activity (PMA)	• Provides funding to programs such as a capability development program (CDP). • Decides programs priorities. • Decides programs hardware and software.
2. Integrated Product Team (IPT)	• Provides lab spaces and network infrastructures to projects. • Decides organizational processes. • Submits CDP. • Manages contracts. • Mitigates program risks.
3. Project Team (PT)	• Manages and executes project requirements. • Participates in critical design review and senior management review. • Provides personnel work status and project metrics to IPT.
4. Business Financial Management (BFM)	• Sends, receives, and processes funding documents. • Provides labor and materials expenditures to the project team.
5. Contractor Unit	• Provides work and services to the project.
6. Software Distribution Unit (SDU)	• Packages and distributes software to trainers, to the DT/OT unit, and to the fleet.
7. Trainer/Fleet users Unit (T/F	• Trains fleet users on how to use fleet software.

Organizational Unit	Responsibilities
Unit)	• Reports software anomalies to the project team.
8. Contracting Unit (CU)	• Provides statements of work and manages contracts.
9. DT/OT Unit	• Performs developmental and operational tests on software.

As illustrated in Figure 13, we define a context diagram convention. We use this diagram convention in the engineering projects throughout this dissertation. The diagram convention is defined as follows: (1) The single arrow-line points from the sending unit to the receiving unit; (2) The description above the single arrow-line represents the flow of information from one unit to another unit; (3) The first number in the parentheses indicates the frequency of the communications and the second number represents the value of importance of the communications. When we measure organizational complexity, we use the context diagram convention to represent the two important relationships (connectedness and interdependence) in terms of the flow of information of the communications between organizational units. These data are available when the project starts.

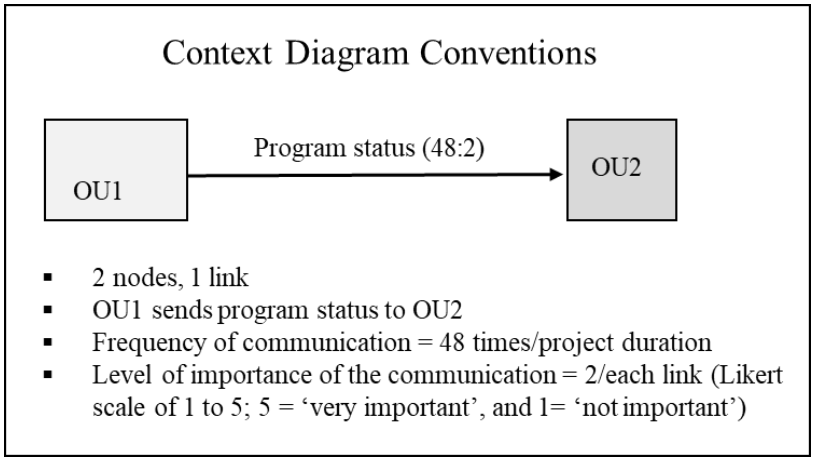

Figure 13. Context diagram conventions used in this dissertation

Figures 14, 15, 16, and 17 represent the nine organizational units (nodes) and their relationships (links) involved in the B1 Project. We use these figures to compute the overall organizational complexity of the B1 Project.

As shown in Figure 14, IPT has three links to PMA. First, IPT sends the program status 36 times to PMA during the first 18 months of the project, and the level of importance of each communication is 3. Second, IPT sends the program funding status 18 times to PMA during the project, and the level of importance of each communication is 3. Third, IPT submits CDPs 1 time to PMA during the first 18 months of the project, asking funding for some additional capabilities of the program, and the level of importance of the communication is 5.

In addition, PMA has five links to the IPT. First, PMA sends CDP decisions 1 time to IPT during the 18-month project, and the level of importance of the communication is 5. Second, PMA provides CDP funding documents 6 times to IPT during the first 18 months, and the level of importance of each communication is 3. Third, PMA sends program priorities and statuses 18 times to IPT, and the level of importance of each communication is 1. Fourth, PMA sends hardware and software requirements 1 time to IPT, and the level of importance of the communication is 5. Fifth, PMA provides the DT/OT report 1 time to IPT during the 18 months project, and the level of importance of the communication is 4.

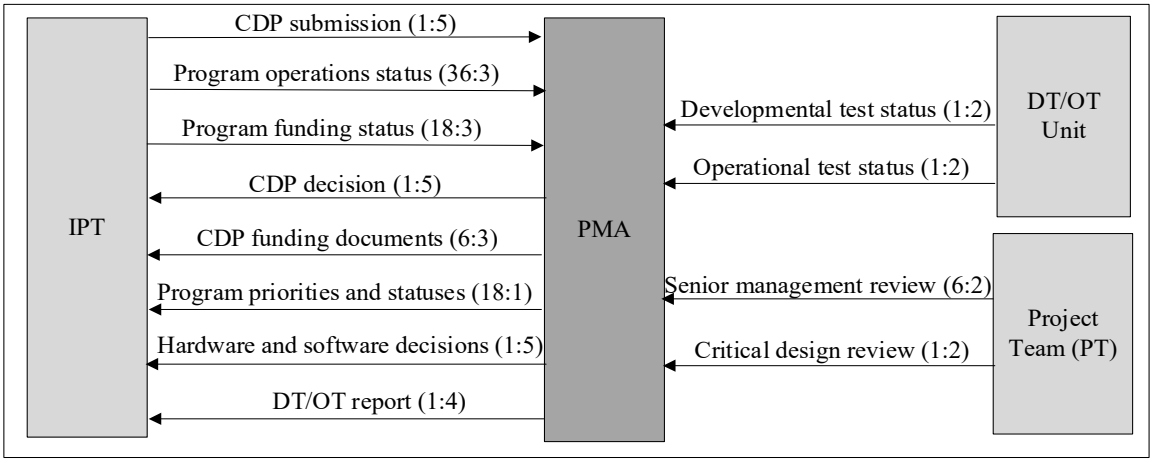

Figure 14. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit

Figures 15, 16, and 17 show the rest of the nodes, links, communication frequencies, and levels of importance of the communications between the organizations of the B1 Project.

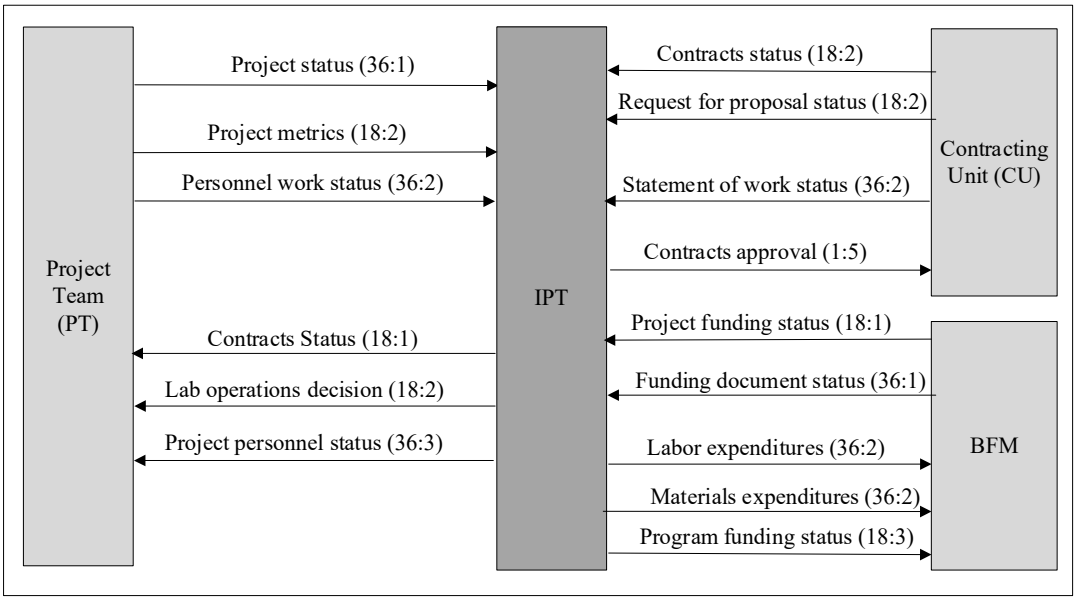

Figure 15. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications

between IPT and PT, between IPT and CU, and between IPT and BFM

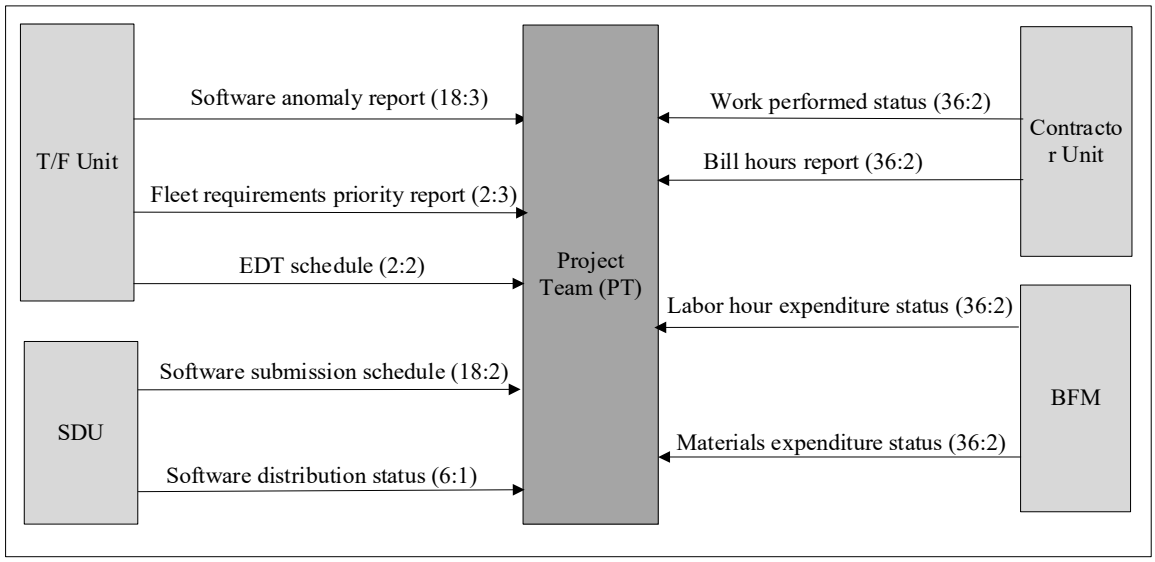

Figure 16. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PT and the T/F unit, between PT and SDU, between PT and BFM, and between PT and the contractor unit

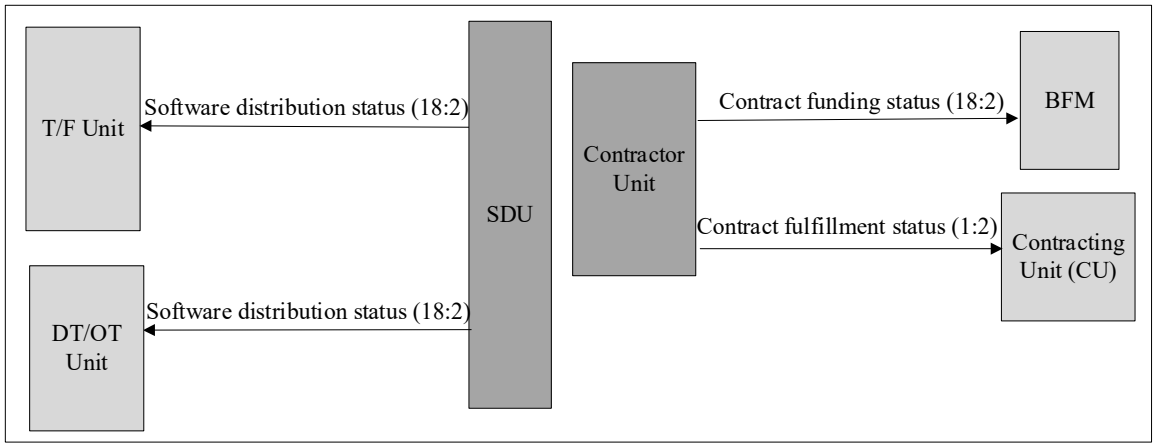

Figure 17. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between SDU and the T/F unit and between SDU and the DT/OT unit as well as the communication frequencies and levels of importance of the communications between the contractor unit and BFM and between the contractor unit and CU

In sum, we identified nine organizational units and their patterns of information flow in the B1 Project. We use the patterns of information flow, along with that information's frequency and importance, to characterize connectedness and interdependence between organizational units. Table 23 shows the N2 chart of the nine organizational units and 40 links in the B1 Project.

Table 23. N2 chart of the B1 Project

	IPT	PMA	Project Team	BFM	Contracting Unit	T/F Unit	DT/OT Unit	Total
1. IPT		3	3	3	1			10
2. PMA	5							5
3. Project Team	3	2						5
4. BFM	2		2					4
5. Contracting Unit	3							3
6. Contractor Unit			2	1	1			4
7. SDU			2			1	1	4
8. T/F Unit			3					3
9. DT/OT Unit		2						2
Total links	13	7	12	4	2	1	1	40

Tables 24 and 25 present the frequencies and levels of importance of the communications in the B1 Project during the 18 months. We use these empirical data to measure organizational complexity between organizational units. The combination of frequency and the level of importance of communications between organizational units can determine how much effect the patterns of the communications have upon project complexity.

Table 24. Frequencies of the communications during the 18 months in the B1 Project

	Communication of information	**Frequency**						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
1. IPT	1. Program operations status		36					

	Communication of information	Frequency						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
	2. Program funding status		18					
	3. CDP submission		1					
	1. Contracts status			18				
	2. Lab operations status			18				
	3. Project personnel status			36				
	1. Labor expenditures				36			
	2. Materials expenditures				36			
	3. Program funding status				18			
	1. Contracts approval					1		
	Subtotal		55	72	90	1		
2. PMA	1. CDP decisions	1						
	2. CDP funding document	6						
	3. Program priorities status	18						
	4. Hardware/software decisions	1						
	5. DT/OT report	1						
	Subtotal	27						
3. Project Team (PT)	1. Project status	36						
	2. Project metrics	18						
	3. Personnel work status	36						
	1. Critical design review status		1					
	2. Senior management review		6					
	Subtotal	90	7					
4. BFM	1. Project funding status	18						
	2. Funding document status	36						
	1. Labor hour expenditure status			36				
	2. Materials expenditure status			36				
	Subtotal	54		72				
5. Contracting Unit (CU)	1. Contracts status	18						
	2. RFP status	18						
	3. Statement of work status	36						
	Subtotal	72						

	Communication of information	Frequency						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
6. Contractor Unit	1. Work performed status			36				
	2. Bill hours report			36				
	1. Contract funding status				18			
	1. Contract fulfillment status					1		
	Subtotal			72	18	1		
7. T/F Unit	1. EDT schedule			2				
	2. Software anomaly report			18				
	3. Fleet requirements priority report			2				
	Subtotal			22				
8. DT/OT Unit	1. DT status		1					
	2. OT status		1					
	Subtotal		2					
9. SDU	1. Software submission schedule			18				
	2. Software distribution status			6				
	1. Software release status						18	18
	Subtotal			24			18	18
	Total	242	64	262	108	2	18	18

Table 25. Levels of importance of the communications in the B1 Project

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
1. IPT	1. Program operations status		3					
	2. Program funding status		3					
	3. CDP submission		5					
	1. Contracts status			1				
	2. Lab operations status			2				
	3. Project personnel status			3				
	1. Labor expenditures				2			
	2. Materials expenditures				2			
	3. Program funding status				3			

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
	1. Contracts approval decisions					5		
2. PMA	1. CDP decisions	5						
	2. CDP funding documents	3						
	3. Program priorities status	1						
	4. Hardware/software decisions	5						
	5. DT/OT report	4						
3. Project Team (PT)	1. Project status	1						
	2. Project metrics	2						
	3. Personnel work status	2						
	1. Critical design review status		2					
	2. Senior management review		2					
4. BFM	1. Project funding status	1						
	2. Funding document status	1						
	1. Labor hour expenditure status			2				
	2. Materials expenditure status			2				
5. Contracting Unit (CU)	1. Contracts status	2						
	2. RFP status	2						
	3. Statement of work status	2						
6. Contractor Unit	1. Work performed status			2				
	2. Bill hours report			2				
	1. Contract funding status				2			
	1. Contract fulfillment status					2		
7. T/F Unit	1. EDT schedule			2				
	2. Software anomaly reporting			3				
	3. Fleet requirements priority status			3				
8. DT/OT Unit	1. DT status		2					
	2. OT status		2					
9. SDU	1. Software submission schedule			2				
	2. Software distribution status			1			2	2

From Equation (3.3) and Tables 24 and 25, we calculate the index function of each organizational unit. Table 26 presents the number of contributions from each organizational unit to the organizational complexity. Over the duration of the development phase, the overall organizational complexity is the sum of all contributions from the organizational units. In this example, we sum up the contributions of organizational complexity from the nine organizational units during the period of 18 months and obtain a

value of 7.71. This means that the organizational complexity of each organizational unit has contributed to some degrees of an adverse effect upon the B1 Project in terms of schedule delay and cost overrun due to high interdependencies between organizational roles and the large number of information flows for communication and decision-making. The result of this example is consistent with the results from previous works found in the related literature [28], [157].

In addition, as discussed in Chapter II, Schwandt's [158] weight factor of a size measure is 0.61. The organizational complexity that is based on the structural size of a project team is 5.49 (the size of a project team, which is 9, times 0.61). In the B1 Project, we have computed the organizational complexity as 7.71, which is about 40% above Schwandt's method [158] of estimating organizational complexity based on the size of an organization. This difference can be explained by the heterogeneity of behaviors (i.e., the patterns and volumes of communication and the levels of importance of the information in the communications) of the nine organizational units.

The uncertainty and ambiguity in terms of the absence or presence of information within and between each organization required to perform the tasks also contribute to the differences of the organizational complexity in the PCMM. Hence, the PCMM and associated methods to measure organizational complexity is compatible with Schwandt's study [158] on organizational complexity in terms of size driver (e.g., the number of organizational units) for systems engineering.

Table 26. Organizational complexity of each organizational unit in the B1 Project

Organizational Unit	***IF***				Total
1. IPT interacts with PMA, PT, BFM, and CU.	PMA: [(3)(36)+(3)(18)+(5)(1)]/[(5)(55)] = 0.607	PT: [(1)(18)+(2)(18)+(3)(36)]/[(5)(72)] = 0.45	BFM: [(2)(36)+(2)(36)+(3)(18)]/[(5)(90)] = 0.44	CU: [(1)(5)]/[(5)(1)] = 1	2.49
2. PMA interacts with IPT.	IPT: [(5)(1)+(3)(6)+(1)(18)+(5)(1)+(4)(1)]/[(5)(27)]=0.37				0.37
3. PT interacts with IPT and PMA.	IPT: [(1)(36)+(2)(18)+(2)(36)]/[(5)(90)] = 0.32	PMA: [(2)(1)+(2)(6)]/[(5)(7)] = 0.4			0.72
4. BFM interacts with IPT and PT.	IPT: [(1)(18)+(1)(36)]/[(5)(54)]= 0.2	PT: [(2)(36)+(2)(36)]/[(5)(72)] = 0.4			0.6
5. CU interacts with IPT.	IPT: [(2)(18)+(2)(18)+(2)(36)]/[(5)(72)] = 0.4				0.4
6. Contractor Unit interacts with PT, BFM, and CU.	PT: [(2)(36)+(2)(36)]/[(5)(72)] = 0.2	BFM: [(2)(18)]/[(5)(18)] = 0.4	CU: [(2)(1)]/[(5)(1)] = 0.4		1
7. T/F Unit interacts with PT.	PT: [(2)(2)+(3)(18)+(3)(2)]/[(5)(22)] = 0.581				0.581
8. DT/OT Unit interacts with PMA.	PMA: [(2)(1)+(2)(1)]/[(5)(2)] = 0.4				0.4
9. SDU interacts with PT, T/F Unit, and DT/OT Unit.	PT: [(2)(18)+(1)(6)]/[(5)(24)] = 0.35	T/F Unit: [(2)(18)]/[(5)(18)] = 0.4	DT/OT Unit: [(2)(18)]/[(5)(18)] = 0.4		1.15
Overall Complexity					7.71

Table 26 shows that IPT has the greatest number of tasks and communications among the organizational units. It has the highest organizational complexity (*IF* = 2.49) of the overall complexity of 7.71 among project organizations. PMA has the least number of tasks and communications, which, in turn, has the lowest organizational complexity (*IF* = 0.37) of the overall complexity among project organizations. The data from this example

show a positive correlation between organizational complexity and the number of organizational units and tasks involved in the engineering project. In short, the PCMM and associated methods to measure organizational complexity have built upon earlier work by Baccarini [14] and Schwandt [158] by identifying the specific individual factors (i.e., the number of interdependent organizational units and the frequency and importance of their communications) that contribute to organizational complexity.

d. Geographical Distribution of Teams Measure

The geographically dispersed of organizations can be challenging for the ongoing relations with business partners at various sites. Because of the differences in time zones, business and cultural norms, native language, and technology used for communication, geographic barriers can affect the project requirements. The causes of complexity mainly arise from lack of system level integration, lack of control over the dispersed teams, the incoherence of the teams, and communication issues.

To illustrate the PCMM and associated methods to measure geographical distribution of teams, we analyze the B1 Project organizational structure in the project plan. The project plan indicates that the 9 organizational units are located in 13 sites and situated in 4 different cities across 2 time zones (Table 27). These data are available when the project starts. Using Equation (3.4), we can calculate the degree of geographical distribution of teams (*GD*) by adding up the number of sites (*s*), locations (*l)*, and time zones (*t*) as shown in Table 27. The result is the *GD* measurement of 19 (4 + 13 + 2). The Allen Curve [115] shows that communication decreases exponentially as distance increases and becomes stable at around eight meters. As the distance increases beyond eight meters between two collaborative teams, there is only a 5% probability of communication once a week among these two teams [119]. As shown in Table 27, PT, the contractor unit, and the DT/OT unit have two to three sites. These sites are located more than eight meters apart. This means that according to Allen [119], these three organizations have a 5% probability to communicate once a week with other organizational units.

In an ideal situation for team communications where we assume all nine organizational units are situated at one site and located in one city in one time zone, the

degree of geographical distribution of teams is calculated as 3 (1 + 1 + 1). Comparing this value with the value of *GD* of the B1 Project, it shows that the B1 Project is about 6.3 times (19 divided by 3) less likely to communicate than in the ideal situation. Nevertheless, a typical Navy software acquisition program with a $5 million annual budget usually have a similar number of organizational units as the B1 Project because the IPT approach to software development usually adopts the functional organizations for a specific purpose of delivering a product for an external or internal customer [197]. Essentially, degrees of collaboration between teams in the B1 Project depend much on the number of locations and sites. Hence, the PCMM and associated methods to measure geographical distribution of teams are reasonable and applicable to a typical Navy software acquisition program that has an annual budget of less than $5 million.

Table 27. Geographical distributions of teams in the B1 Project

	Number of Sites	**Location**	**Time Zone**
PMA	1	East Coast city	Eastern time
IPT	1	Southwest city 1	Pacific time
PT	3	Southwest city 1	Pacific time
BFM	1	Southwest city 1	Pacific time
CU	1	Southwest city 1	Pacific time
Contractor Unit	2	Southwest city 1	Pacific time
T/F Unit	1	Northwest city	Pacific time
DT/OT Unit	2	East Coast city and Southwest city 2	Eastern time and Pacific time
SDU	1	Southwest city 1	Pacific time
Total	13	4	2

e. Requirements Volatility Measure

Requirements volatility indicates the level of risk associated with the huge changes in system development effort, cost, and time as well as the quality of the product which may result in project delay or possible project failure.

As explained in Chapter II, the requirements volatility measure is typically less than 1 [169], [170]. A number greater than one means that a greater number of requirements have added, modified, or deleted than the number of requirements originally planned in the release. In this case, the project has been in the development phase for 18 months. The requirements lead has been tracking the requirements changes and provided the requirements traceability report during this period. We obtain the data for this measure for a given baseline from a requirements baseline report. As discussed in Chapter II, high requirements volatility indicates uncertainty of the system and poor understanding of environment and system [161], [166].

From the B1 Project profile (Table 16) and Equation (3.5), we calculate the value of requirements volatility in the B1 Project. The overall requirements volatility of the B1 Project (i = total of 12 baselines measured by 18 months) is calculated as follows:

$$RVM = \frac{CR}{BR} = \frac{528}{1946} = 0.27.$$

The computed requirements volatility of the B1 Project is similar to the requirements volatility of 0.36 in Latif et al.'s case study [195]. Compared to Stark et al.'s studies [155] of 0.64 (the first two years of releases) and 0.3 (the last two years of releases) for a typical small size of software project (annual budget of $5 million or less), the calculated requirements volatility of the B1 Project also has a similar result. Table 27 shows the B1 Project requirements metrics during the 18-month development phase. During an 18-month period, the B1 Project had 528 change requests (151 new, 1 modification, and 376 deletions) and 1,721 requirements (Table 28). In Figure 18, we provide a plot of requirements volatility against the baseline version number with baseline 0 being the start of this new project. From Figure 26, the requirement changes are fast in baseline 4, indicating either uncertainty of the system or poor understanding of the environment and system. The requirement changes slows down after baseline 5, indicating that the number of changes is getting smaller. As stated in Equation (3.5), the rate of requirements volatility depends on the total number of requirements changes. Furthermore, as shown in Figure 18, requirement changes tend to decrease toward the end of the development life cycle. This finding confirms Nurmuliani et al.'s [166] and Nurmuliani et al.'s [196] hypotheses that

requirements volatility tends to decrease toward the end of the development life cycle and has a great impact on the development effort. Hence, the PCMM and associated methods to measure requirements volatility are acceptable to a typical Navy software acquisition program that has an annual budget of less than $5 million.

Table 28. B1 Project requirements metrics during the development phase (18 months)

	Requirement (CR_i)			BR = 1,946 requirements	
Baseline (i)	New	Modification	Deletion	Total # of requirements	Requirements volatility = $(CR_i)/(BR)$
0				1,946	0
1	11			1,957	0
2	69		8	2,018	0.04
3	31			2,049	0.02
4			309	1,740	0.16
5	4		28	1,716	0.02
6	20	1	5	1,731	0.01
7	5			1736	0
8	6		3	1,739	0
9			22	1,717	0.01
10	1			1,718	0
11			1	1,717	0
12	4			1,721	0
Total	151	1	376		

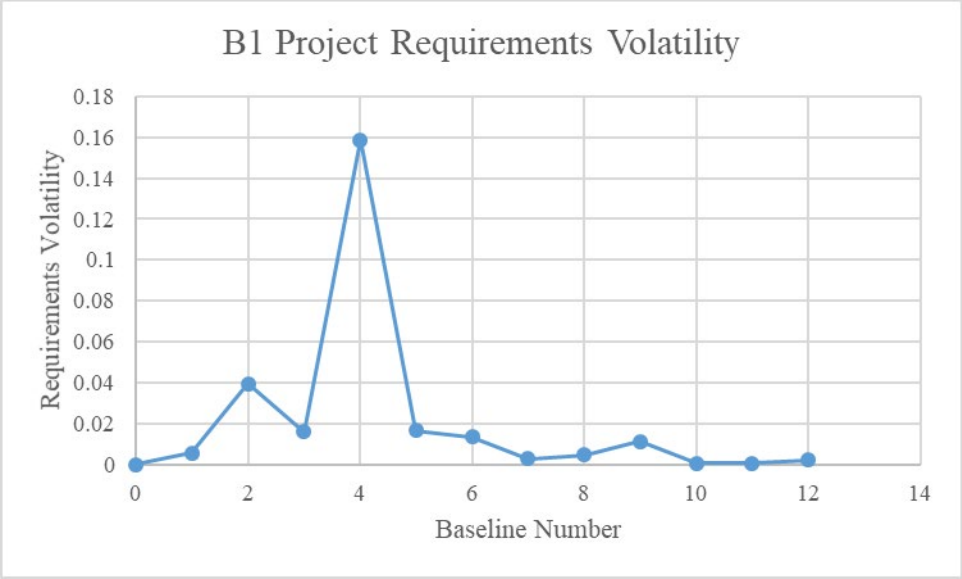

Figure 18. B1 Project requirements volatility

f. Number of Different Job Position Types

Using the material surveyed in the literature review of Chapter II, we know that diversity in types is often associated with complexity. For example, different types of job positions in interdependent organizational units are likely to foster differences in job assignments and outcomes, and, therefore, increased complexity. Thus, we model the number of different job position types in organizational units as diversity in types.

To demonstrate measurement of the number of different job position types in organizations, we analyze the B1 Project organizational structure in the project plan. The project plan indicates 44 different types of job positions. Table 29 lists the 44 different types of job positions in the 9 organizations. Table 30 shows the responsibilities for each job position type in the B1 Project. The data shown in Tables 29 and 30 are available when the project starts.

Table 29. Job position types in the B1 Project

	Job Position Type	
1. PMA	a. Program manager b. Executive officer c. Logistics lead d. Business operations manager	
2. IPT	a. IPT site lead b. Military lead c. Office manager d. Chief systems engineer e. Program-related engineering lead	f. Lead technologist g. International programs lead h. Process improvement lead i. Fleet help desk specialist j. Training manager
3. PT	a. Product lead b. Project manager c. Systems engineer d. Design lead e. Requirements lead f. Programmer	g. Software installer h. Test lead i. Configuration manager j. Product documentation lead k. Standards compliance lead l. Tester
4. BFM	a. BFM lead b. Financial analyst c. Financial technician	
5. CU	a. Contracting officer b. Contracting analyst c. Contract specialist	
6. Contractor Unit	a. Contract representative b. Site manager c. Technical team lead	
7. T/F Unit	a. Fleet liaison b. Trainer c. Fleet representative	
8. DT/OT Unit	a. Lead software engineer b. Information assurance specialist c. Technical analyst	
9. SDU	a. Program analyst b. Product integrity lead c. Systems engineer lead	

Table 30. Responsibilities for job position types in the B1 Project

Job Position Type	Responsibility
1. Program manager	Lead, plan, budget, and manage the acquisition and execution of programs, interface with international partners and foreign military sales (FMS) customers.
2. Executive officer	Coordinate and decide program's funding and priorities.
3. Logistics lead	Coordinate and distribute hardware to various programs at the PMA level.
4. Business operations manager	Perform supervisory responsibilities associated with the strategic planning, risk management, and administrative and management activities for various programs.
5. IPT site lead	Direct, plan, budget, and manage the execution of programs at the IPT level.
6. Military lead	Serve as IPT site lead from the military chain of command.
7. Office manager	Provide credit card buys, maintain training records and create reports for various data calls.
8. Chief systems engineer	Review, evaluate, coordinate, and monitor programs at the IPT level.
9. Program-related engineering lead	Coordinate and allocate funding for program-related engineering and program-related logistics.
10. Lead technologist	Coordinate and execute capability development across multiple product lines and on solution development with other competencies.
11. International programs lead	Collaborate and coordinate with the defense science and technology (DST) group and foreign partners to develop new capabilities. Provide support to FMS customers.
12. Process improvement lead	Plan, review, and execute standards compliance and process improvement policies established by the IPT.

Job Position Type	Responsibility
13. Fleet help desk specialist	Provide help desk support to fleet users by answering phone calls and responding to emails from the fleet.
14. Training manager	Plan, schedule, coordinate, and conduct process improvement training.
15. Product lead	Direct and lead development of project plans including resource requirements, timelines, priorities, and budget impact.
16. Project manager	Coordinate, assess, and monitor software development schedules, contracts, budgets, and personnel of the B1 Project.
17. Systems engineer	Analyze requirements, functional interfaces, and functional architectures of the B1 software project.
18. Design lead	Provide technical guidance to the software development team and is responsible for the software architectures of the B1 Project.
19. Requirements lead	Develop, review, analyze, and maintain software requirements for the B1 Project.
20. Programmer	Develop code and perform software integration activities.
21. Software installer	Develop software installation packages for deployment.
22. Test lead	Plan and manage all test-related functions including test strategy, test plan development, test execution and reporting, development of personnel skills, and improvement of processes.
23. Configuration manager	Perform configuration management duties in support of software releases. Develop and maintain build scripts, operational procedures, and internal documentation.
24. Product documentation lead	Develop software user's guide and technical manuals.
25. Standards compliance lead	Assess, review, and monitor process compliance in each program.

Job Position Type	Responsibility
26. Tester	Test software and record software defects.
27. BFM lead	Monitor programs for project management requirements. Perform financial analysis and reporting.
28. Financial analyst	Provide requirement instructions related to funding documents.
29. Financial technician	Prepare, process, and reconcile funding documents and related issues. Respond to data calls and cost estimating.
30. Contracting officer	Coordinate, implement, and monitor the contract compliance program. Provide technical advice and assistance in all areas of contracted support services.
31. Contracting analyst	Provide advice and assistance to the contracting officer and program managers in cost and schedule contractual management, ensuring proper interpretations of reporting requirements.
32. Contract specialist	Negotiate sole source contracts and modifications. Provide guidance to technical personnel involved in the development of contract packages.
33. Contract representative	Maintain records of performance schedules and work progress reports. Monitor contractor performance and/or negotiate settlements.
34. Site manager	Lead, plan, execute, and conduct analyses concerning all aspects of services performed by contract.
35. Technical team lead	Serve as a supervisor of technical teams for contracting work.
36. Fleet liaison	Plan, coordinate, report, and execute fleet requirements. Analyze and resolve fleet-related issues.
37. Trainer	Conduct operational software training to fleet users. Test released software and report software anomalies to product lead.

Job Position Type	Responsibility
38. Fleet representative	Work closely with project managers and trainers in reviewing fleet requirements for the program.
39. Lead software engineer	Evaluate the effectiveness of DT/OT software function in addressing operational and information security requirements.
40. Information assurance specialist	Evaluate, assess, analyze, and test software systems to ensure compliance with information assurance policies, instructions, and directives.
41. Technical analyst	Assist lead software engineer to evaluate, assess, analyze, and test DT/OT software.
42. Program analyst	Interface with customers from multiple technical teams and provide services required by customers.
43. Product integrity lead	Coordinate and distribute software to customers from multiple technical teams according to SDU release schedule and distribution list.
44. Systems engineer lead	Coordinate, assess, and monitor software release schedule for a specific program within the SDU.

Using Equation (3.6), we calculated *JPT* and obtain 44. The 44 job position types interact and exchange information in many different ways that contribute to project complexity. Some of the job positions support several projects and share the labor cost among them. As compared to a typical Navy software acquisition program approach, the IPT usually adopts functional roles similar to those found in a typical Navy software acquisition program, as shown in the Table 29 [197]. These roles execute the project plan and ensure that lower-level processes occur and products are created. Hence, the PCMM and associated methods to measure the number of different job position types are sensible to a typical Navy software acquisition program that has an annual budget of less than $5 million.

As shown in step 8 of Figure 6, we have applied the PCMM and created a complexity profile for the B1 Project. In the next section, we present the complexity profile of the B1 Project and provide an understanding of the impact that a given complexity value is likely to have on engineered systems and projects.

4. B1 Project Complexity Profile

Using the PCMM, we have calculated the complexity values of the B1 Project. Table 31 presents the complexity profile of the B1 Project.

Table 31. Complexity profile of the B1 Project

Complexity Measure	PCMM Value	Results from previous works and conclusions drawn based on the IPT-related literature
Number of Personnel required for the development effort (people)	18.85 people per year	20 people per year for projects with an annual budget of $3 million [151 –[153]
Defect density (defects per KLOC)	0.32	≤ 3 [160]
Organizational Complexity (*IF*)	7.71	5.49 based on Schwandt's study [158]
Geographical distribution of teams (*GD*)	19	Reasonable when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]
Requirements volatility (*RVM*)	0.27	< 1 [162]
Number of different job position types (*JPT)*	44	Sensible when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]

As indicated in step 9 of Figure 6, we analyze the complexity profile of the B1 Project. In regard to the measurement of the personnel required for the development effort,

the Project Management Institute (PMI) provides a guideline for determining the size of the project (e.g., small project < $5M, medium project < $50M, large project < $500M, and mega project > $500M) [159]. Based on an annual budget of $3 million, the project manager could fund approximately 20 people ($3 mil/$145,932). The PCMM estimates 18.85 people. The PMBOK [185] says that project reserves are 5% to 10% of the estimated cost. Project reserves cover residual risks in the project. Thus, the PCMM shows that we need annual funding for 20.74 people (18.85 times 1.1) or $3.026 million. Based on the original annual budget of $3 million, the PCMM shows a risk for $26,000 shortfalls in the annual budget.

In the defect density measure, the software industry standard is to keep defects per KLOC under 3 [161]. In the B1 Project, the PCMM estimates 0.32. This means that the PCMM shows low risk ($0 < DD < 3$) in terms of the stability of the software releases.

In the organizational complexity measure, the literature from previous works [25], [157] supports the hypothesis that high interdependencies between organizational roles and a large number of information flows for communication and decision-making contributes to organizational complexity. Based on Schwandt's approach [158], we can estimate the organizational complexity as 5.49 (9 organizational units times Schwandt's [158] weight factor of 0.61). Note that an *IF* value of less than 0.5 in each organizational unit in the project will generally indicate low risk related to organizational complexity [158]. In the B1 Project, the PCMM estimates 7.71 or an average *IF* value of 0.86 (7.71 / 9). This means that the PCMM indicates medium risk ($0.5 < IF < 0.9$) related to organizational complexity [158].

The Allen curve [119] phenomenon supports the complexity measure of geographical distribution of teams. In an ideal work environment, *GD* is 3 (one site, one location, and one time zone) to maintain a high probability of communication once a week for team collaboration. The PCMM estimates *GD* of 19 based on the data from the B1 project plan. Nevertheless, when comparing to the IPT approach in software development, a typical Navy software acquisition program with an annual budget of less than $5 million usually has a similar number of organizational units as the B1 Project [197]. Hence, *GD* of 19 is a reasonable estimate for a typical Navy software program. This means that the

PCMM indicates low to medium risk ($3 < GD < 20$) related to the complexity of team collaboration.

In the requirements volatility measure, the best practice in the software industry is to keep the rate of changes at less than one for reducing the risk of requirements uncertainty and requirements creep [162]. The PCMM estimates *RVM* of 0.27 based on the empirical data from a requirements metric report of the B1 Project. This means that the PCMM shows low risk ($0 < RVM < 1$) related to requirements volatility in the B1 Project.

Diversity in types is often associated with complexity. In Chapters I and II, we explained that different job position types in interdependent organizational units are likely to have differences in job assignments and outcomes, and, therefore, increased complexity. The PCMM estimates degrees of different job position types as 44. Based on a typical Navy software acquisition program approach to software development, the IPT usually adopts functional roles similar to those found in the B1 Project [197]. This means that the PCMM indicates low risk ($JPT < 45$) related to the complexity of different job position types.

In sum, based on the B1 Project empirical data, we draw a conclusion that the PCMM and associated methods to measure project complexity are more reasonable compared to several studies in the literature [151]–[153], [158], [160], [162], [197]. The higher the value of each complexity measure from the PCMM, the more complex the project, and vice versa. The complexity values of the B1 Project obtained from the PCMM provide program managers a level of understanding in terms of risk and dependency areas in the B1 Project.

In general, for any system development project, it is more difficult or costly to correct an error discovered late in the life cycle. The PCMM program managers can use a project's complexity values computed from the PCMM as a guide for project comparisons, to plan for project cost and schedule and to identify relevant risk areas in software projects. We need to determine the scores and associated levels of complexity of six complexity metrics in the PCMM to identify the overall B1 Project complexity level.

First, to determine the score and associated level of complexity of the personnel required metric, we use the project size of $5 million annual budget from the PMI [159] as a guide and $145,932 yearly salary of a software developer [152][153] to calculate to the

number of people that can be funded annually for the project. It is 34 people ($5 million / $145,932). We divided 34 into 5 ranges of scores. We use a Likert scale [32] of 1 to 5 and map the score of the personnel required metric to the associated level of complexity as shown in Table 32.

Table 32. The score and associated level of complexity of the measure of the number of personnel required for the development effort

Metric	PCMM value	Number of people can be funded per year (PMI)	Score	Complexity Level
1. Number of personnel required for the development effort (people)		1 to 7	1	Simple
		8 to 15	2	Complicated
	20.74	16 to 27	3	Low complexity
		28 to 34	4	Moderate complexity
		> 34 or unknown	5	High complexity

Second, to determine the score and associated complexity level of the defect density metric, we use the software industry standard of keeping the number of defects per KLOC less than 3 [160] as a guide and divide this value of 3 into 5 ranges of scores. We use a Likert scale [32] of 1 to 5 to map the score of the defect density metric to the associated level of complexity as shown in Table 33.

Table 33. The score and associated level of complexity of the measure of defect density

Metric	PCMM value	Defects / KLOC (software industry standard)	Score	Complexity Level
2. Defect density (defects / KLOC)	0.32	0 to 0.5	1	Simple
		0.51 to 1.1	2	Complicated
		1.2 to 2	3	Low complexity
		2.1 to 3	4	Moderate complexity
		> 3 or unknown	5	High complexity

Third, to determine the score and associated level of complexity of the organizational complexity metric, we use Schwandt's study [158] of 5.49 as a reference number for low complexity. We use a Likert scale [32] of 1 to 5 to map the score of the organizational complexity metric to the associated level of complexity as shown in Table 34.

Table 34. The score and associated level of complexity of the measure of organizational complexity

Metric	PCMM value	*IF* (Schwandt's study [158])	Score	Complexity Level
3. Organizational complexity		0 to 1.8	1	Simple
		1.9 to 3.7	2	Complicated
		3.8 to 5.5	3	Low complexity
	7.71	5.6 to 8.1	4	Moderate complexity
		More than 8.1 or unknown	5	High complexity

Fourth, to determine the score and associated level of complexity of the geographical distribution of teams metric, we use the Allen curve [119] as a guide for conducting optimum team communications (one site, one city, and one time zone) and use the IPT functional roles [197] of 19 as a reference number for low complexity. We also use

a Likert scale [32] of 1 to 5 to map the score of the geographical distribution of teams metric to the associated level of complexity as shown in Table 35.

Table 35. The score and associated level of complexity of the measure of geographical distribution of teams

Metric	PCMM value	*GD* (guide from the Allen curve [119] and the IPT roles [197])	Score	Complexity Level
4. Geographical distribution of teams		1 to 3	1	Simple
		4 to 10	2	Complicated
	19	11 to 19	3	Low complexity
		20 to 40	4	Moderate complexity
		More than 40 or unknown	5	High complexity

Fifth, to determine the score and associated level of complexity of the requirements volatility metric, we use the best practice in the software industry of keeping the rate of changes at less than one [162] as a threshold value for moderate complexity. We use a Likert scale [32] of 1 to 5 to map the score of the requirements volatility metric to the associated level of complexity as shown in Table 36.

Table 36. The score and associated level of complexity of the measure of requirements volatility

Metric	PCMM value	*RVM* (best practice in the software industry [162])	Score	Complexity Level
5. Requirements volatility		0 to 0.2	1	Simple
	0.27	0.21 to 0.42	2	Complicated
		0.43 to 0.66	3	Low complexity
		0.67 to 1	4	Moderate complexity
		More than 1 or unknown	5	High complexity

Sixth, to determine the score of the PCMM for the number of different job position types, we use the IPT job positions [197] of 44 as a guide for low complexity. We use a Likert scale [32] of 1 to 5 to map the score of the number of different job position types metric to the associated level of complexity as shown in Table 37.

Table 37. The score and associated level of complexity of the measure of the number of different job position types

Metric	PCMM value	*JPT* (guide from the IPT roles [197])	Score	Complexity Level
6. Number of different job position types		1 to 15	1	Simple
		16 to 30	2	Complicated
	44	31 to 45	3	Low complexity
		46 to 65	4	Moderate complexity
		More than 65 or unknown	5	High complexity

To determine the overall complexity level for the B1 Project, we compute the average of the complexity scores of the six measures in the PCMM as shown in Table 38. Hence, the overall complexity level of the B1 Project complexity is 2.67 (low complexity), as shown in Table 38.

Table 38. The B1 Project complexity level

Metric	PCMM value	Complexity Score	Complexity Level
1. Number of personnel required for the development effort (people)	20.74	3	Low
2. Defect density (defects / KLOC)	0.32	1	Simple
3. Organizational complexity	7.71	4	Moderate
4. Geographical distribution of teams	19	3	Low
5. Requirements volatility	0.27	2	Complicated
6. Number of different job position types	44	3	Low
Overall B1 Project complexity level		2.67	Low

C. SENSITIVITY ANALYSIS

The B1 Project in Section B illustrated that the PCMM is both useful and effective. Nevertheless, to further show the practical use of the PCMM, we perform a sensitivity analysis on the B1 Project. The goal of sensitivity analysis is to determine the range of input parameters where the project can achieve a desirable outcome. The approach is to change the technical parameters of the B1 Project and observe the value of the decision variables. We use sensitivity analysis to demonstrate the validity of the PCMM and show the robustness and reliability of the analysis used by the model.

In this section, we perform a sensitivity analysis on the measure of the number of personnel required for the development effort. We initially make a 30% change to the annual budget of the B1 Project. Then we increase the number of CRs and requirements by 30% as shown in Table 39.

Table 39. A 30% increase to the three B1 Project inputs

Project Property	Original Value	Original Value increased by 30% for sensitivity analysis
Estimated Yearly Budget	$3 million	$.3.9 million
Number of Requirements	1,721 requirements	2,237 requirements
Number of Change Requests (CRs)	528 CRs (151 new, 1 modification, 376 deletions)	686 CRs (196 new, 1 modification, 489 deletions)
Number of "nominal" requirements	344 requirements	447 requirements
Number of "easy" requirements	1,377 requirements	1,790 requirements

Based on these changes, we estimate the number of personnel required for the development effort. Using an annual budget of $3.9 million from Table 39 and a yearly salary of $145,932 for a software developer, we could hire 26.72 people ($3.9 mil / $145,932). Table 40 shows the results of COSYSMO size for the B1 Project.

Table 40. Calculation of COSYSMO size

COSYSMO size's driver [24]	B1 Project software size for sensitivity analysis
Requirements	(1,790 easy requirements)(0.5) = 895 (447 nominal requirements)(1) = 447 Total = 895 + 447 = 1,342
Interfaces	(2 easy interfaces)(1.7) = 3.4
Algorithms	(1 nominal algorithm)(6.5) = 6.5
Operational Scenarios (use cases)	(32 easy use cases)(9.8) = 313.6
Total COSYSMO size [24] = requirements + interfaces + algorithms + user cases	1,342 + 3.4 + 6.5 + 313.6 = 1,665.5

We keep the same value of the effort weight factor of 0.77 [24] as shown in Table 19. We also keep the values of the COSYSMO calibration factors *A* as 0.325 [24] and the default value of *E* as 1 [24]. Finally, we use Equation (3.1) to calculate the amount of person-month required for the B1 Project and present the results in Table 41.

Table 41. The B1 Project development effort

B1 Project development effort	Value
Effort required to build the system = *PM* = (calibration factors)(effort factor)(software size)	(0.325)(0.77)(1,665.5) = 416.79 person-month
B1 Project development effort = (effort required to build the system) / (duration of the development phase)	416.79 / 18 = 23.15 people

As shown in Table 41, the PCMM estimates 23.15 people. Compared to the original value of 18.85 people, the percentage of change in the personnel required measure is 22.81% (23.15 / 18.85 – 1 x 100%). This is expected because we made a 30% change to three input parameters of the B1 Project and the PCMM increases by 22.81%. Hence, the sensitivity value of the PCMM is 0.24 (1 - 0.2281/ 0.3).

Next, we increase the annual budget by another addition of 10%. We also increase the number of CRs and requirements by an addition of 10% as shown in Table 42.

Table 42. An additional increase of 10% to the three B1 Project inputs

Project Property	Original Value	Original Value increased by 10% for sensitivity analysis
Estimated Yearly Budget	$4.29 million	$4.29 million
Number of Requirements	2,461 requirements	2,461 requirements
Number of Change Requests (CRs)	686 CRs (196 new, 1 modification, 489 deletions)	755 CRs (216 new, 1 modification, 538 deletions)
Number of "nominal" requirements	447 requirements	492 requirements
Number of "easy" requirements	1,790 requirements	1,969 requirements

Once again, using the annual budget of $4.29 million shown in Table 42 and an annual salary of $145,932 for a software developer, we could hire 29.39 people ($4.29 mil / $145,932). Tables 43 shows the results of the COSYSMO size [24] and Table 44 shows the personnel required for the B1 Project.

Table 43. Calculation of COSYSMO size [24]

COSYSMO size's driver [24]	B1 Project software size for sensitivity analysis
Requirements	(1,969 easy requirements)(0.5) = 984.5 (492 nominal requirements)(1) = 492 Total = 984.5 + 492 = 1,476.5
Interfaces	(2 easy interfaces)(1.7) = 3.4
Algorithms	(1 nominal algorithm)(6.5) = 6.5
Operational Scenarios (use cases)	(32 easy use cases)(9.8) = 313.6
Total COSYSMO size [24] = requirements + interfaces + algorithms + user cases	1,476.5 + 3.4 + 6.5 + 313.6 = 1,800

Table 44. The B1 Project development effort

B1 Project development effort	Value
Effort required to build the system = *PM* = (calibration factors)(effort factor)(software size)	(0.325)(0.77)(1,800) = 450.45 person-month
B1 Project development effort = (effort required to build the system) / (duration of the development phase)	450.45 / 18 = 25.02 people

As shown in Table 44, the PCMM estimates 25.02 people. Compared to the original value of 23.15 people before the 10% changes, the percentage of change in the personnel required measure is 8.08% (25.02 / 23.15 – 1 x 100%). This is expected because we made an additional change of 10% to the three input parameters and the PCMM increases by 8.08%. Thus, the sensitivity value of the PCMM is 0.19 (1 – 0.0808 / 0.1), which is less than 0.24 as expected. Hence, the sensitivity value of the PCMM is stable, dropping from 0.24 to 0.19.

Regarding the requirements volatility measure, we increase the number of change requests by 30%. Using Tables 16 and 39 as well as Equation (3.5), we calculated the overall requirements volatility of 0.35 (686 / 1,946), which is increased by 30% compared to the original value of 0.27. Hence, the sensitivity value of the PCMM is 0.29 (0.35 / 0.27 – 1).

Next, with the additional increase of 10% as shown in Table 42, the calculated overall requirements volatility is 0.388 (755 / 1,946), which is increased by 10% as expected compared to the original value of 0.35. Hence, the sensitivity value of the PCMM is 0.1 (1 - 0.388 / 0.35), which is less than 0.29 as expected. Thus, the sensitivity value of the PCMM is stable, dropping from 0.29 to 0.1.

In sum, we performed a sensitivity analysis on both the number of personnel required for the development effort and the requirements volatility of the PCMM. The analysis shows that regarding both measurements, the PCMM is sensitive to the changes in the technical parameters of the B1 Project. At the same time, the PCMM is stable within

the ranges of the technical parameters where the project does sustain the different levels of complexity. We will perform a sensitivity analysis on the defect density measurement using the project data in Chapter IV.

D. COMPLEXITY REDUCTION AND MITIGATION

Project managers can reduce project complexity by simplify their project structures and communications platform. These changes can potentially provide cost saving while strengthening the execution of project tasks. Project managers use complexity reduction as a management tool to streamline business processes and optimize information systems with project objectives.

In step 10 of Figure 6, we suggest ways to reduce or mitigate complexity. For example, in a system development program, we can reduce or mitigate complexity of the measure of defect density by reducing or eliminating redundant designs and processes [147]. One way is by consolidating multiple functional requirements into a common platform or framework to improve design efficiency. The effect of the fact that the improvement of design efficiency in software simplify the code, making it easy to maintain and to debug in a unit test. As a result, software developers can potentially reduce the number of small and common mistakes in the program control logic. Therefore, the code is more robust to execute and has fewer dependencies in the operational environment. The number of software defects will decrease over time, which results in a reduction of defect density.

To connect these complexity reduction techniques to the PCMM model, we need to explain the relevance of these techniques in regard to the PCMM metrics. Let's describe the application of these techniques to each measure of the PCMM in order.

First, to reduce the number of personnel required for the development effort, the program manager can reduce the size of the project by limiting the project scope through planning early in the development cycle. By dividing the number of system requirements into several increments of software blocks, the program manager can reduce the number of major interfaces, critical algorithms, and operational scenarios in each block. The requirements lead can perform requirements prototyping when end user requirements are

volatile. In addition, the program manager and software lead can reduce the cost driver EM_i of the project by performing the following:

- Increase the level of requirements understanding to very high by conducting technical interchange meetings and requirements peer reviews.
- Reduce the technical risk of the project to very low by performing design prototyping.
- Increase the process capability to very high by conducting training on the CMMI process [191] and process improvement.

Furthermore, the software lead can consolidate multiple functional requirements into a common platform or framework to improve design efficiency [198]. Moreover, the program manager can reduce or eliminate redundant processes to optimize process efficiency by streamlining information systems with business objectives. Hence, we can reduce project complexity by reducing the size of development effort.

Second, in terms of reducing defect density, the software lead in the project can reduce the number of defects (ND_i) in each baseline ($KLOC$) by setting coding standards and conducting code walk-through procedures that developers should abide by. The software lead can show developers software architectures and examples of how to develop loosely coupled code, making code easier to maintain. In addition, the program manager can standardize on one IT platform that provides configuration controls of code releases. Furthermore, the software lead can leverage a modular design of code and reuse that code to improve design efficiency and limit the amount of code change within a software unit.

Third, to reduce organizational complexity of the project, the program manager can reduce the communication frequencies and the levels of importance of the communications among organizational [147]. The program manager can set a priority and a baseline in project reporting as well as focus on critical decisions and customers' needs by aligning information systems with business objectives. In addition, the program manager can clarify organizational roles and decision-making processes to best serve the project teams while also streamline on business processes and information systems.

Fourth, to reduce the geographical distribution of teams, the program manager can reduce the number of locations (l), time zones (t), and sites (s) of each organizational unit in the project by performing the following:

- Reduce project activities that take place in multiple locations.
- Change the problem definition or approach in such possible ways that reduces complexity of project activities.

As shown in Table 27, we have identified 13 sites (s =13), 4 cities (l = 4), and 2 time zones (t = 2) in the B1 Project. To reduce *GD*, the program manager can consolidate and rearrange the project activities to reduce the number of geographical locations and different time zones during the project. In addition, the program manager can consolidate and rearrange the lab spaces and seating arrangements between developers to fit the number of sites. Thus, the combination of reducing the number of sites and the number of project activities in multiple locations can reduce project complexity and improve team collaboration.

Fifth, to reduce the requirements volatility of the project, the program manager can reduce the number of change requirements (CR_i) in each baseline of the release (*BR*) by performing requirements prototyping and implementing a formal change approval process. Furthermore, the program manager can freeze the project scope early in the development cycle and postpone change requests that are not immediately needed by the customers, raising the hurdles for new capability requirements and other expansion activities that add complexity cost. The program manager also needs to limit the addition of new requirements late in the development life cycle. For example, in Table 31, a requirements volatility measure of 0.27 means that the risk is low regarding requirements creep and the insufficient understanding of the software requirements and use cases. To keep the project risk level low during the early development cycle, program managers can schedule weekly requirements reviews, design reviews, and test case peer reviews to clarify requirements understanding among the requirements lead, design lead, test lead, and developers.

Six, to reduce the number of different job position types in the project, the program manager can combine or reduce the number of reporting tasks and review activities in each

organizational unit as well as increase the competency level and control the maturity of a project. In Table 31, we have 44 different job position types (*JPM* = 44) that must take place to execute the work. This reflects many engagements and interactions between each job position that might exchange information. To reduce project complexity, the program manager can clarify the organizational roles, reduce levels of management, and improve spans of control of each organization.

E. CHAPTER SUMMARY

The PCMM presented in this chapter is derived from literature related to the sources of complexity and the four major types of project complexity presented in Chapter II. The indicative metrics of the four types of complexity of the SoI presented in Table 15 provide a systematic way to measure the degree of complexity of a software project. We can draw parallels of the PCMM to 4 of the 7 building blocks of complexity—numerosity, connectedness, interdependence, and diversity. Furthermore, literature about types of complexity and project complexity from Chapter II provides evidence on the validity of the PCMM and the research methodology presented in this chapter. In the next chapter, we present three software engineering projects and apply the PCMM and associated methods to calculate the overall degree of complexity of each program. The engineering projects (named in this dissertation as B4, B6, and B8 programs) are intended to demonstrate the validity and relevancy of the PCMM through in-depth analysis and results.

IV. PROJECT DATA TO DEMONSTRATE THE VALIDITY OF THE PCMM

> Given a problem to solve, figure out how to do it once, and then do it the same way each time.
>
> —Ali A. Minai, Dan Braha, and Yaneer Bar-Yam [79], "Complex Engineered Systems: A New Paradigm"

As noted in the Chapter II literature review, conflicting requirements, combined problem elements, diverse and demanding stakeholders, and multiple constraints appear to increase the overall complexity of engineering projects. This realization justifies the need for creating a complexity profile that consists of multiple complexity measures for project risk assessment. The objective of this chapter is to demonstrate the validity of the PCMM for the purpose of computing complexity by examining three engineering projects (the software programs B4, B6, and B8) and computing their complexity values. We compare the complexity values of the three engineering projects to the values of the industry common practices in the literature. The output of the PCMM is a value that signifies a project's complexity, which systems engineers can then use to predict, analyze, and understand patterns of communications and activities in systems development. This value might serve as a proxy for resources or time requirements for developing system architecture. The project manager can use the project's complexity profile as supporting evidence to present to senior management and IPT for introducing changes in the project management strategy and for developing potential approaches in complexity reduction. In addition, because complexity contributes to project risk, higher values returned by the PCMM imply that we need to allocate additional time and effort to development, integration, testing, and maintenance.

To confirm the usefulness of the PCMM, we need to ask the following questions:

1. How does the complexity values from the PCMM compare to the values of the industry common practices?

2. Does the PCMM accurately describe the complexity properties of real-world systems?

3. Does the PCMM work as a prescription for complexity analysis in real-world systems?

4. Does the PCMM help systems engineers identify, understand, and assess the impacts a given complexity value is likely to have on real-world systems?

Throughout this chapter, we keep these questions in mind when applying the PCMM to compute the complexity values and develop a complexity profile for each use case. To validate the PCCM's complexity measures, we compare them to the values output by other methods in previous studies and the industry common practice noted in Chapter II. We also perform a complexity reduction analysis for the B6 Project. Figure 19 shows an overview of the configuration management (CM) workflow of the Navy software programs.

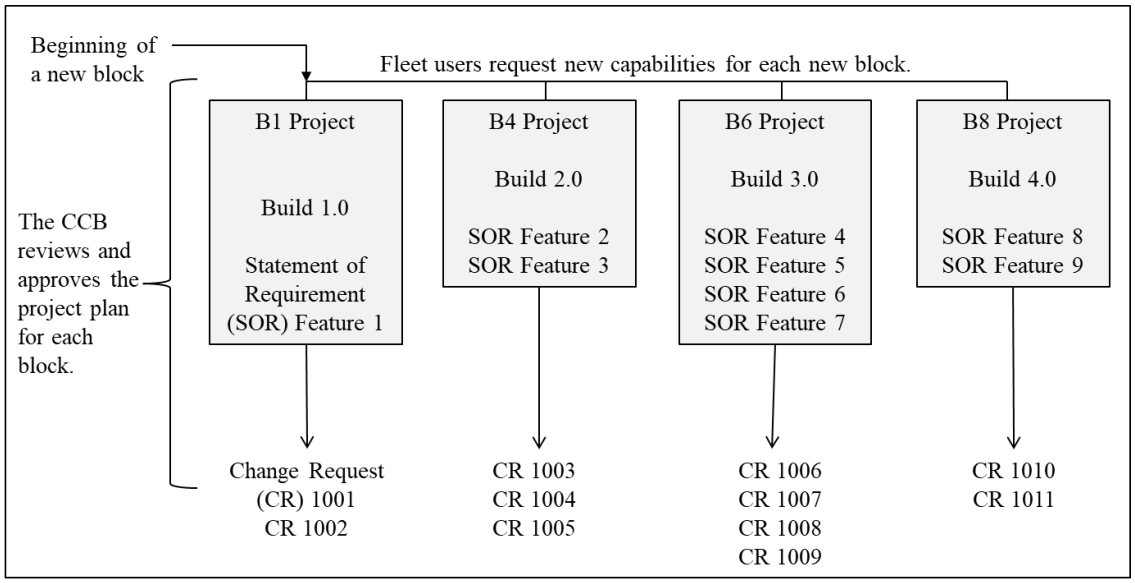

Figure 19. CM workflow of the Navy software programs

The data collected for the three Navy projects (B4, B6, and B8) are from the internal software projects that are supported by the IPT organizations in the related command. The data are collected from various project documents such as the project plan, the test plan, test reports, LOC reports, requirements traceability reports, software design description documents, interface design documents, and use case documents.

In the B4, B6, and B8 Projects, we analyze the empirical project data and determine the following information:

- Annual budget of the project,
- Project duration for that block,
- Number of requirements and change requests listed in the requirements traceability reports,
- Number of use cases listed in the SDD documents,
- Number of interfaces required (obtained from the SDD documents),
- LOC of each release (obtained from the LOC reports), and
- Number of defects and number of test hours conducted for each baseline of the release (obtained from the test reports).

A. B4 WINDOWS-BASED DATABASE SOFTWARE PROGRAM

In this use case, the Navy B4 software project used an evolutionary development method in which the program delivers the software in increments. The overall project is broken up into separate phases which may be representative of the different software builds. Each software build completes a subset of requirements. The project contains the following phases, according to [22], [200]:

1. Creating requirements,
2. Analyzing requirements,
3. Designing software,
4. Writing code and conducting unit test,

5. Integration testing,

6. System integration testing, and

7. Deployment of software.

The purpose of this project is to develop a Windows database application for fleet users to use as a tool for mission planning and data analysis.

Figure 20 depicts an overview of the Navy B4 Project, which includes organizations, project teams, management teams, a project plan, and software requirement specifications (SRS) as well as databases and elements of both hardware and software. In addition, the project has IT systems supporting ad hoc audits of new requirements, project reviews, and configuration management. Test teams and organizations such as DT, OT, trainers, and fleet representatives are responsible for testing the system.

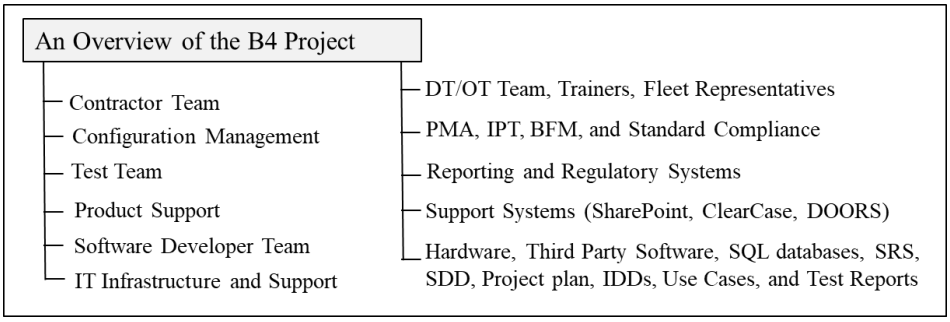

Figure 20. A context diagram of the B4 Project

Figure 9 of Chapter III shows the five steps of the complexity study method (CSM). We use the CSM and apply it to the B4 Project. In step 1 of the CSM, we use the in-house repository system (SharePoint) to obtain the SRS, interface design description (IDD), use case documents, software design document (SDD), project plan, requirements traceability report (RTR), and test plan of the B4 Project. We review SRS, IDD, RTR, and project plan to develop the B4 Project profile.

The B4 Project plan identifies 23 software work products and 18 deliverables that include the project documents. It shows an organizational chart that has 20 team members

in the project. Table A-1 of Appendix A lists the organizations and their responsibilities. In addition, Table A-2 of Appendix A presents the PMA, IPT, and PT that are involved in milestone reviews and status reviews during the 24-month development phase.

The B4 Project plan indicates that the C# object-oriented programming language was used to develop the software application. The IDD document lists four external interfaces. Developers noted in the IDD document that one external interface was determined as nominal to implement because it has 18 classes and several methods to support query and retrieval of data. The B4 test plan lists 32 test procedures that link to 32 use cases in the development phase. Each test procedure provides test steps that verify requirements in each use case. Since the project started in April of 2017, the empirical data of change requests during the 24-month development phase were available. The RTR listed 803 change requests that include 356 new change requests, 18 modifications, and 429 deletions during the 24-month period. The project started with a baseline of 2,067 requirements. At the end of the 24-month period, it had 1,994 requirements.

The level of complexity to implement each requirement depends on how well the requirement is written, how easily it is to track the sources of the requirement, and whether any requirements extend over to other requirements [24]. From the interviews of several program managers of previous programs, some requirements are considered easy because these requirements have been implemented successfully before. According to the previous program manager, these requirements are straightforward and usually take less than a week to implement. Some requirements are considered nominal because they are sophisticated and can take up to 4 weeks to finish. Some requirements are difficult to implement and largely extend over to other requirements [24].

In the case of the B4 Project, the project manager reviews and analyzes the project data of the B1 Project from the previous acquisition program and obtains the level of complexity and the amount of time it took to implement each requirement. However, the dataset of the B1 Project from the previous acquisition program is not complete. Nevertheless, the project manager decides there is no requirement in the difficult category for the B4 Project. Thus, the project manager decides applying the 80–20 rule that is based on the interviews of program managers of previous programs and determines that 80% of

requirements will take less than 1 week to implement while 20% of requirements will take between 2 to 4 weeks. Hence, in the B4 Project, the program manager determines that 1,595 requirements (80% of requirements) will be easy to implement while 399 requirements (20% of requirements) will be nominal to implement.

Requirements understanding refers to the level of familiarity of the system requirements by the development team [24]. Based on his or her work experiences, the project manager assesses the level of familiarity of the system requirements of the development team. The project plan indicated that requirements understanding of the software development team was high.

Technical risk is characterized as the possibility of requiring more SE effort due to immaturity or obsolescence of the technology implemented into the system [24]. This is based on a subjective assessment of the system from the project manager. The project plan listed technical risk as nominal (technology proven on pilot projects).

Process capability refers to the ability of the project team to follow defined processes with a certain degree of effectiveness [24]. The Capability Maturity Model Integration (CMMI) [191] is one of many published process models that is used in rating process capability.

The IPT is a CMMI level 3 (i.e., defined processes) organization. Therefore, the B4 Project follows CMMI processes at the start of the project. The project plan stated that the process capability of the project team was nominal (managed SE process, activities driven by customers' and stakeholders' needs).

External interfaces are defined as logical boundaries between functions which provide handshaking and exchange messages via certain protocols [24]. The IDD listed four external interfaces. The complexity of interfaces is based on a subjective assessment from the software developers. The IDD indicated that three interfaces were easy (simple establishing a communication channel between entities and exchange messages) to implement and one interface was nominal (moderate sophistication use of protocol for setting a communication channel between entities), which requires more engineering effort.

Algorithms represent newly defined functions that require some mathematical methods in order to achieve the system performance requirements [24]. The SDD and project plan identified four critical algorithms. Software developers rated two algorithms as easy (simple data and timing not an issue) and the other two algorithms as nominal (relational data and nested structure with decision logic) based on their experiences.

Use cases capture the system's functionalities [24]. The use case documents and test plan showed 27 use cases in order to validate the system performance. The software developers rated all 27 use cases as easy (well defined system operational scenario).

With the information from the project plan, IDD, use case documents, SDD, test plan, build reports, and requirements metrics report (RTR), we follow step 2 of the CSM shown in Figure 9 and develop the B4 Project profile, as shown in Table 45.

Table 45. B4 Project profile. Adapted from [24].

Project Property	Value	Source of Information
Yearly Budget	$3.3 million	Project plan
Team Size	20	Project plan
Duration of the Development Phase	24 months	Project plan
Baseline Requirement (BR)	2,067 requirements at project start	RTR
Number of requirements at the end of the project	1,994 requirements	RTR
Number of Change Requests (CRs)	803 CRs (356 new, 18 modifications, and 429 deletions)	RTR
Number of "nominal" requirements	399 requirements (20%)	Applying the 80–20 rule that is based on the interview of program managers of previous programs
Number of "easy" requirements	1,595 requirements (80%)	Applying the 80–20 rule that is based on the interview of program managers of previous programs

Project Property	Value	Source of Information
Understanding of requirements [24]	high	Project plan
Technical Risk	nominal	Project plan
Process Capability [24]	nominal	Project plan (CMMI [191] level 3)
Number of External Interfaces	4 (3 “easy” and 1 “nominal”)	IDD
Number of Critical Algorithms [24]	4 (2 “easy” and 2 “nominal”)	Software design document (SDD) and project plan
Number of use cases	27 (all “easy”)	Use case documents and test plan
Number of lines of code (LOC)	870,559	Build report (SLOC count)

We follow step 3 of the CSM shown in Figure 9 and compare the B4 project profile with previous projects based on historical data. In this case, we compare the B4 project profile with the B1 Project profile described in Chapter III because the B4 Project is the next block of software that adds more capabilities to the B1 Project. Hence, the B4 Project profile is similar to the B1 Project profile in terms of the number of requirements. In step 4 of the CSM shown in Figure 9, we apply the PCMM and estimate the B4 Project complexity as shown in the next section.

1. Measure of the Number of Personnel Required for the B4 Project

We use the data from the B4 Project profile shown in Table 45 to compute and obtain a software size value of 1,490.3 as shown in Table A-3 of Appendix A. In addition, using the COSYSMO [24] weight factors of cost drivers shown in Table 17 of Chapter III, we calculate and obtain the effort weight factor of 0.77 as shown in Table A-4 of Appendix A.

Finally, we use Equation (3.1) to calculate the number of person-month required (*PM*) in the B4 Project [24]. We divided the *PM* value of 372.95 person-month by 24 months and obtain 15.54 people as shown in Table A-5 of Appendix A. Adding the project

reserves of 10% to cover residual risks in the project [185], we determine that the B4 Project needs annual funding for 17.09 people (15.54 times 1.1) or $2.494 million (17.09 times $145,932). In this case, the number of personnel required for the development effort estimated by the PCMM is reasonable and well within 20% of the projected personnel required as specified in the B4 Project plan.

2. Defect Density Measure

We need test data to measure defect density. In this case, the test data is available because the B4 Project started in April of 2017 and has completed the development phase. Testers have evaluated the software. Tables A-6 and A-7 of Appendix A show the B4 Project monthly test report, taken from the empirical project data. The plots shown in Figures 21 and 22 reflect the data in Table A-7 of Appendix A.

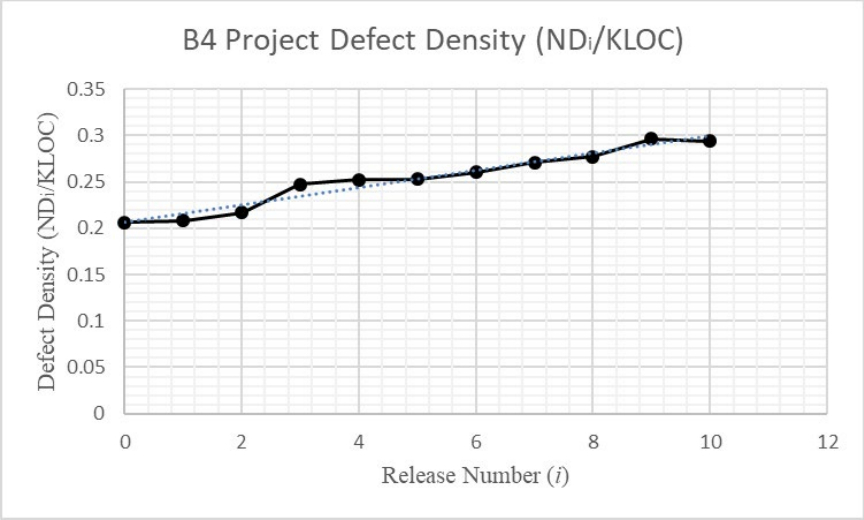

Figure 21. Plot of defect density of the B4 Project

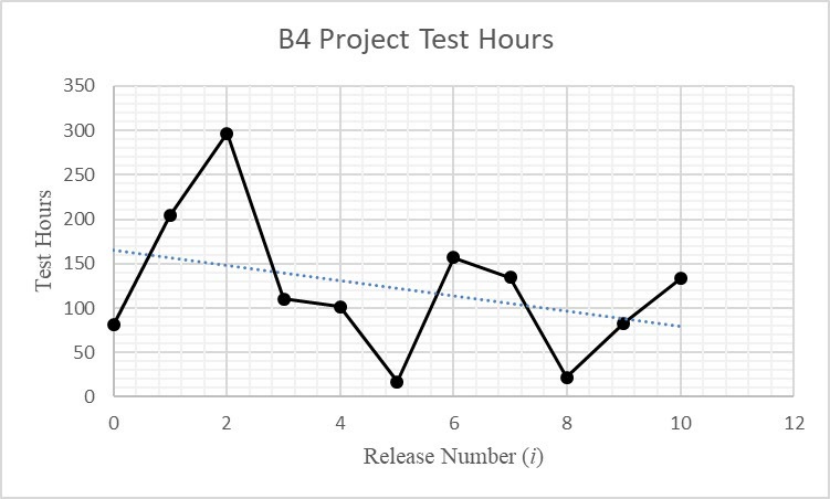

Figure 22. Test Hours of the Project B4

As seen in Figure 21, the defect density ($ND_i/KLOC$) is approximately constant throughout the development cycle. The ND_i//KLOC increased slightly at the end of the development cycle because testers discovered more bugs before the release of the software. The trend line of test hours is downward as illustrated in Figure 22. Testers have to balance their priorities between increasing their test time and shifting their time away from testing and focusing on test metric reporting and updating test procedures. As shown in Table A-7 of Appendix A, the project defect density is 0.299 defects per KLOC, which is within the industry standard limit of less than 3 defects per KLOC [160].

In general, when test hours are increased during the development phase in a software project, the defect density also increases because more defects are discovered. When the test hours decrease significantly, this may indicate that testers have focused on writing test procedures and have not tested the software enough to close the unresolved defects. Program managers should use the defect density report as an indicator of project performance.

3. Organizational Complexity Measure

The B4 Project plan identifies nine organizational units, and their responsibilities are showed in Table A-1 of Appendix A. We characterize connectedness and

interdependence among organizational units by the patterns of information flow, along with the information's frequency and importance.

Figure A-1 of Appendix A identifies the nodes, links, frequencies, and levels of importance of the communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit during the 24-month development phase. Figures A-2, A-3, and A-4 of Appendix A show the context diagrams of the rest of the nodes, links, communication frequencies, and levels of importance of the communications between the organizations of the B4 Project. We use Figures A-1, A-2, A-3, and A-4 of Appendix A to generate the N2 chart as shown in Table 46. The N2 chart represents nine organizational units and 40 links in the B4 Project.

Table 46. N2 chart of the B4 Project

	IPT	PMA	Project Team	BFM	Contracting Unit	T/F Unit	DT/ OT Unit	Total
IPT		3	3	3	1			10
PMA	5							5
Project Team	3	2						5
BFM	2		2					4
Contracting Unit	3							3
Contractor Unit			2	1	1			4
SDU			2			1	1	4
DT/OT Unit		2						2
T/F Unit			3					3
Total links								40

From Equation (3.3) and Tables A-8 and A-9 of Appendix A, we calculate the index function of each organizational unit that is associated with the links to another organizational unit. Table A-10 of Appendix A presents the organizational complexity of each organizational unit and the total organizational complexity of 7.66.

This means that the organizational complexity of each organizational unit has to some degree contributed to the effect upon the B4 Project due to the large number of information flows for communication and decision-making. In this use case, we have

computed the organizational complexity as 7.66, which is similar to the result derived from Schwandt's method [158] of estimating organizational complexity of 5.49 (the size of a project team, which is 9, times the weight factor of a size measure, which is 0.61). This difference can be explained by the heterogeneity of behaviors (i.e., the patterns and volumes of communication and the levels of importance of the information in the communications) of the nine organizational units. The ambiguity in terms of the absence or presence of information within and between each organization required to perform the tasks also contribute to the differences of the organizational complexity in the PCMM.

As shown in Table A-10 of Appendix A, IPT has the greatest number of tasks and communications in the project, which, in turn, have contributed to the highest organizational complexity (*IF* = 2.52) among project organizations. CU has the fewest number of tasks and communications, which, in turn, have contributed to the lowest organizational complexity (*IF* = 0.2) among project organizations.

4. Geographical Distribution of Teams Measure

From the B4 Project plan, we identify 13 sites situated in 4 cities and across 2 time zones. Using Equation (3.4), the degree of geographical distribution of teams (*GD*) is determined by adding up the number of sites (*s*), locations (*l)*, and time zones (*t*) as shown in Table A-11 of Appendix A. In this case, we compute *GD* and obtain 19 (13 + 4 + 2). According to Allen [119], a *GD* value of greater than 3 means that there is a less than 5% probability that each of the 9 organizational units communicates once a week with other organizational units. This means that collaboration between teams is infrequent, which can contribute to a higher risk of schedule delay, cost overrun, or project failure. However, a typical Navy software acquisition program with a $5 million annual budget usually adopts the IPT approach to software development that includes these functional organizations for a specific purpose of delivering a product to internal customers and fleet users [197]. These organizations in the B4 Project adjusted their project schedules and held weekly status meeting for overcoming the barrier of geographical distribution of teams to achieve the project deliverables.

5. Requirements Volatility Measure

Using Equation (3.5) and the B4 Project profile shown in Table 45, we calculate the value of requirements volatility in the B4 Project. The overall requirements volatility measure (*RVM*) of the B4 Project is as follows:

$$RVM = \frac{CR}{BR} = \frac{803}{2067} = 0.388$$

where *CR* denotes the number of changes in requirements (new requirements, modifications of requirements, and deletions of requirements) during the 24-month period, and *BR* denotes the baseline of the software release, which is measured by the number of requirements.

We obtain the data of requirements baseline 0 through baseline 22 from the B4 Project requirements metric report and present them in Table A-12 of Appendix A.

As shown in Figure 23, we plot the requirements volatility against the baseline version number with baseline 0 being the start of this new project. The requirements volatility (CR_i / BR) is high in baselines 1, 4, 10, and 16, indicating either uncertainty of the software system or poor understanding of the environment and system. Uncertainty generally is a contributing factor to schedule delay due to indecisive project planning decisions. The requirements volatility (CR_i / BR) eases after baseline 16, indicating that the number of requirement changes decreases. Furthermore, as indicated in Figure 23, requirement changes tend to decrease toward the end of the development life cycle.

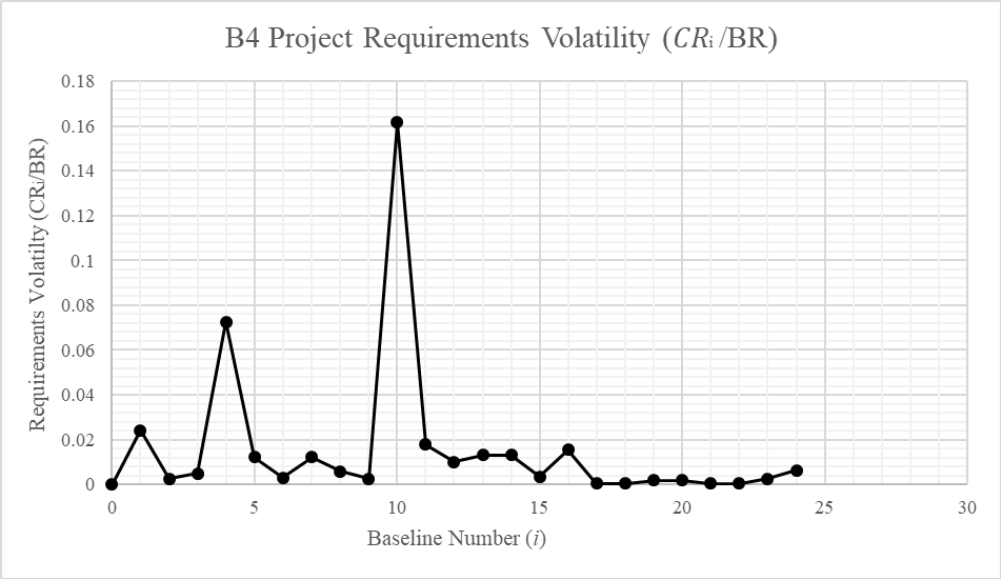

Figure 23. Requirements volatility of the B4 Project

6. Number of Different Job Position Types

The B4 Project plan lists 44 job position types, and we present them in Table A-13 of Appendix A. The B4 Project plan also describes the responsibilities of the 44 job position types, and we list them in Table A-14 of Appendix A. Using Equation (3.6), we calculate *JPT* by counting all job position types and the value is 44.

People who worked in these different types of job positions interact, exchange, and process information in many different ways that contribute to project complexity. Some of the people in these job positions support several projects, and the labor cost are shared between the projects.

7. B4 Project Complexity Profile

Using the PCMM, we have calculated the complexity values of the B4 Project. Table 47 presents the complexity profile of the B4 Project. This complexity profile is the six measures of the PCMM. The six measures represent a combination of four different types of complexity (structural complexity, organizational complexity, temporal complexity, and technological complexity).

Table 47. Complexity profile of the B4 Project

Complexity Measure	PCMM Value	Results from previous works and conclusions drawn based on the IPT-related literature
Number of Personnel required for the development effort (people)	17.09 people per year	22 people per year based on projects with an annual budget of $3.3 million [151]–[153]
Defect density (defects per KLOC)	0.299	≤ 3 [160]
Organizational Complexity (*IF*)	7.66	5.49 based on Schwandt's study [158]
Geographical distribution of teams (*GD*)	19	Reasonable when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]
Requirements volatility (*RVM*)	0.388	< 1 [162]
Number of different job position types (*JPT)*	44	Sensible and consistent when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]

We follow step 5 of the CSM shown in Figure 9 and analyze the results to identify potential opportunities to reduce project risks. The purpose of this analysis is to determine the validity of the PCMM.

First, regarding the measurement of the number of personnel required for the development effort, the project manager can determine the number of people and the right mix of different employees to work on the project. An adequate number of personnel to maintain the project schedules and handle the project tasks as well as popup assignments reduces project risk. Based on the estimated annual project budget shown in the project

plan, we could fund 22.6 people ($3.3 million / $145,932) yearly for the B4 Project. Nevertheless, the project plan listed 20 people and allocated some funding for project reserves. The PCMM estimated 17.09 people. This indicates very low risk related to the size of the team because the measurement of the personnel required by the PCMM is less than 20 people. This analysis shows that the PCMM is fairly consistent and accurate in estimating the level of complexity in terms of the number of people required for the annual budget compared to the B4 Project plan. Note that the project manager uses the project plan as an initial plan for project cost, schedule, and deliverables when the project starts.

Second, because the development phase was completed in this case, the test data were available to compute the defect density. When the B4 Project started in April of 2017, the project manager used the results of the B1 Project and estimated the defect density (0.32) for the B4 Project because the B1 Project was the previous block of software of the B4 Project as shown in Figure 19. The PCMM estimated 0.299 defects per KLOC based on the empirical data. This indicates very low risk related to the stability of the released software because the defect density measurement is less than 3 defects per KLOC as specified by the software industry standard [160]. The PCMM is reasonable and consistent in estimating the level of complexity based on the test data of the B4 Project.

Third, in the organizational complexity measurement, the PCMM estimated a value of 7.66 for the organizational complexity. This indicates low risk related to the patterns and volumes of communication among the organizations of the project team because the average weight factor of the 9 organizations is 0.85 (7.66 divided by 9), which is close to 1 and greater than the Schwandt's [158] weight factor of 0.61. The result of this measurement is consistent with the results from previous works found in the relevant literature [28], [157].

Fourth, the PCMM estimated a value of 19 for the measurement of the geographical distribution of teams. This indicates medium risk related to the degrees of collaboration between teams because this measurement is greater than 3 (one site, one location, and one time zone) for an ideal team environment in maintaining a high probability of communication once a week [119]. Nevertheless, the PCMM is sensible and consistent with a typical Navy software acquisition program because the IPT approach to software

development usually adopts the functional organizations shown in Table A-1, Appendix A, for the specific purpose of delivering a product for internal and external customers [197].

Fifth, because the development phase was completed in this case, the requirements traceability report was available to compute the requirements volatility. When the B4 Project started in April of 2017, the project manager used the results of the B1 Project and estimated the requirements volatility (0.27) for the B4 Project because change requests for additional requirements were added to the requirements baseline of the B1 Project as illustrated in Figure 19. The PCMM estimated the requirements volatility measurement of 0.388 based on the empirical data. This indicates low risk related to requirements creep and the understanding of the software requirements as well as use cases because the *RVM* is less than 1 as indicated in the literature [162]. The PCMM is consistent and reasonable in estimating the level of complexity based on the requirements traceability report of the B4 Project.

Sixth, in the measurement of the number of different job position types, the PCMM estimated 44 different job position types based on the B4 Project plan. This indicates low risk related to the execution of tasks, job assignments, organizational roles, and management control because the IPT often adopts functional roles similar to those found in a typical Navy software acquisition program, as shown in A-14, Appendix A. The PCMM is consistent in estimating the level of complexity in terms of the number of different job position types based on the B4 Project plan.

The overall complexity level of the B4 Project is computed as the average of the six complexity scores as shown in Table-21 of Appendix A. As presented in Table A-21 of Appendix A, the overall complexity score of the B4 project is 2.67 and that the complexity level is low.

In regard to the B4 Project risk, we use Table A-22 of Appendix A to score the six metrics in Tables A-15, A-16, A-17, A-18, A-19, and A-20 of Appendix A. We rate the risk level of the project based on degree of impact in terms of cost, schedule, and performance on the project, as shown in Table A-23 of Appendix A. The overall risk level

of the B4 Project is calculated as the average of the risk scores of the six PCMM measures, which is 1.167 as shown in Table A-24 of Appendix A. Thus, the overall risk level of the B4 Project is low as presented in Table A-24 of Appendix A. As illustrated in Table A-23 of Appendix A, a low-risk project means a cost increase of less than 10% to the initial project cost.

Table 48 shows the complexity level and risk level of each metric in the B4 Project.

Table 48. Complexity level of the B4 Project

Metric	Complexity Score	Complexity Level	Risk Level
1. Number of personnel required for the development effort	3	Low	Low
2. Defect density	1	Simple	Low
3. Organizational Complexity	4	Moderate	Medium
4. Geographical distribution of teams	3	Low	Low
5. Requirements volatility	2	Complicated	Low
6. Number of different job position types	3	Low	Low
Overall complexity level of the B4 Project	2.67	Low	Low

In sum, the PCMM helps project managers determine areas of possible concern and provide unique practices to manage complex projects. Based on the PCMM assessment of the B4 Project, one area of concern is the organizational complexity, which is indicated as a medium risk. This means a potential cost increase of 10% to 20% to the project cost as

shown in Table A-23. The project manager may need to spend some time determining how the team will distribute and retrieve project information. Overall, the assessment of low complexity and low risk of the B4 Project demonstrated the consistency of the PCMM because the B4 Project was a low-risk project. According to the B4 Project manager and the IPT senior management review, the B4 Project delivered the software on schedule and well within budget. The software met the performance requirements based on the operational test report.

B. B6 WINDOWS-BASED DATABASE SOFTWARE PROGRAM

The purpose of the B6 Project is to develop a Windows database application for fleet users to use as a tool for mission planning and data analysis. The B6 Project is the next software block of the B4 Project, and it uses an evolutionary development method through which the project delivers the software in increments. The B6 software program provides more capabilities than the B4 software program by implementing additional requirements requested from the fleet. The B6 Project is broken up into separate phases which are representative of the different software builds. Each software build completes a subset of requirements and contains phases that are similar to those of the B4 Project. Figure 24 shows an overview of the B6 Project, which includes organizations, project teams, management teams, a project plan, and SRS as well as databases and elements of both hardware and software. In addition, the project has IT systems supporting ad hoc audits of new requirements, project reviews, and configuration management. Test teams and organizations such as DT, OT, trainers, and fleet representatives are responsible for testing the system.

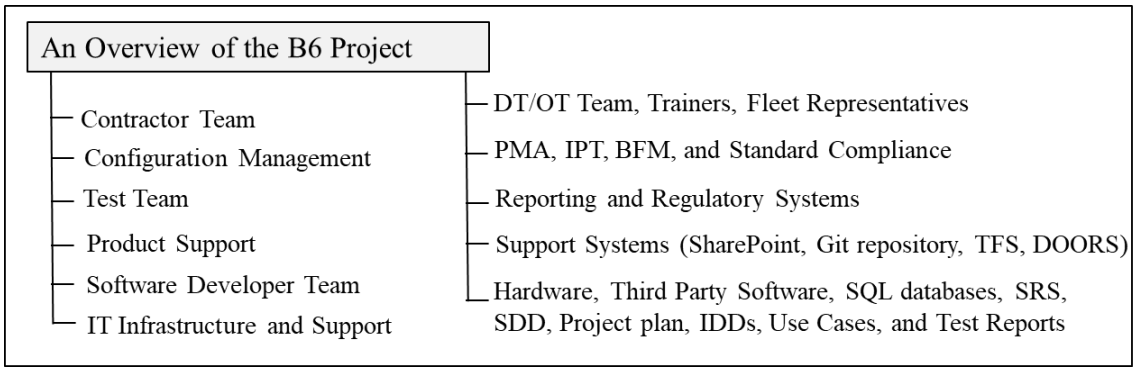

Figure 24. A context diagram of the B6 Project

The B6 Project plan identifies 21 software work products and 21 deliverables that include project documents. It shows an organizational chart that has 34 team members in the project. The B6 Project has nine organizational units that are identical to those in the B4 Project as shown in Table A-1 of Appendix A. The responsibilities of these nine organizational units are listed in Table A-1 of Appendix A. In addition, Table B-1 of Appendix B presents the PMA, IPT, and PT that are involved in milestone reviews and status reviews during the first 28-month development phase.

The B6 Project plan indicates that the integrated development environment is Microsoft Windows 10 and the C# object-oriented programming language. The IDD documents and test procedures list 17 external interfaces. Software developers determined and noted in the IDD documents that 4 external interfaces were difficult to implement because these interfaces have several classes that require some methods to perform lengthy computations for analyzing the data. Developers also determined that 6 interfaces were nominal to implement because these interfaces have classes that require several methods to perform calculations for data mapping. The B6 test plan lists 39 test procedures that link to 39 use cases in the development phase. Each test procedure provides test steps that verify requirements in each use case. Since the project started in February of 2017, data related to change requests during the 28-month development phase were available. The RTR listed 2,725 change requests that included 1,586 new change requests, 286 modifications, and 853 deletions during the 28-month period. The project started with a baseline of 2,089 requirements. At the end of the 28-month period, it had 2,822 requirements.

The project manager follows the approach of the B4 Project and determines that 80% of requirements will take less than 1 week to implement while 20% of requirements will take between 2 to 4 weeks. Hence, the program manager determines that 2,258 requirements (80% of requirements) will be easy to implement and concludes that 282 requirements (10% of requirements) will be nominal to implement. The remaining 10% requirements (282 requirements) will be difficult to implement.

Similar to the B4 Project plan, the B6 Project plan indicated that requirements understanding of the software development team was high and technical risk was nominal. In addition, the process capability of the project team was nominal.

Software developers noted in the SDD that three algorithms were easy (simple data manipulation and timing not an issue) to develop. Two algorithms required nominal effort from the developers to implement because of the relational data and nested structure with decision logic. Two algorithms were difficult to create because of the complicated relational data and several nested structures with decision logic. Developers determined and noted in the use case documents that 29 use cases were easy to generate. Six use cases required a nominal amount of time from the developers to create the operational scenarios. Four use cases were difficult to develop because of complicated scenarios that involve multiple assets and these assets have interactions among each other.

With the information from the project plan, IDDs, use case documents, SDD, test plan, test procedures, build reports, and RTR, we create the B6 Project profile as shown in Table 49.

Table 49. B6 Project profile

Project Property	Value	Source of Information
Estimated Yearly Budget	$5.3 million	Project plan
Team Size	34	Project plan
Duration of the Development Phase	28 months	Project plan
Baseline Requirement (BR)	2,089 requirements at project start	RTR

Project Property	Value	Source of Information
Number of Requirements at the end of the project	2,822 requirements	RTR
Number of Change Requests (CRs)	2,725 CRs (1,586 new, 286 modifications, 853 deletions)	RTR
Number of "difficult" requirements	282 requirements (10%)	Allocating 10% of requirements as "difficult" based on the interview of the project manager of the B4 Project
Number of "nominal" requirements	282 requirements (10%)	Allocating 10% of requirements as "nominal" based on the interview of the project manager of the B4 Project
Number of "easy" requirements	2,258 requirements (80%)	Applying the 80–20 rule that is based on the interview of the project manager of the B4 Project
Understanding of requirements [24]	high	Project plan
Technical Risk [24]	nominal	Project plan
Process Capability [24]	nominal	Project plan (CMMI [191] level 3)
Number of External Interfaces	17 (7 "easy," 6 "nominal," and 4 "difficult")	IDD and test procedures
Number of Critical Algorithms [24]	7 (3 "easy," 2 "nominal," and 2 "difficult")	Software design document (SDD) and project plan
Number of use cases	39 (29 "easy," 6 "nominal," and 4 "difficult")	Use case documents and test plan
Number of lines of code (LOC)	790,487	Build report (SLOC count)

With the project profile shown in Table 49, we apply the PCMM and estimate the B6 Project complexity in the next section.

1. Measure of the Number of Personnel Required for the B6 Project

We use the data from the B6 Project profile shown in Table 49 to calculate and obtain a software size value of 1,490.3, as shown in Table B-2 of Appendix B. In addition, we use the COSYSMO weight factors of cost drivers [24] shown in Table 17 of Chapter III to calculate and obtain the effort weight factor of 0.77, as shown in Table B-3 of Appendix B.

Finally, we use Equation (3.1) to calculate the amount of person-month required (*PM*) [24] in the B6 Project and obtain 31.89 people as shown in Table B-4 of Appendix B. Adding the project reserves of 10% to cover residual risks in the project [174], the B6 Project needs annual funding for 35.08 people (31.89 times 1.1) or $5.119 million (35.08 times $145,932). In this case, the number of personnel required for the development effort estimated by the PCMM is reasonable and not exceeding 5% of the projected personnel required as specified in the B6 Project plan.

2. Defect Density Measure

Since the project started in February of 2017, we have test data to measure defect density. Table B-5 of Appendix B shows the B6 Project monthly test report, taken from the empirical project data. The plots shown in Figures 25 and 26 reflect the data in Table B-5 of Appendix B.

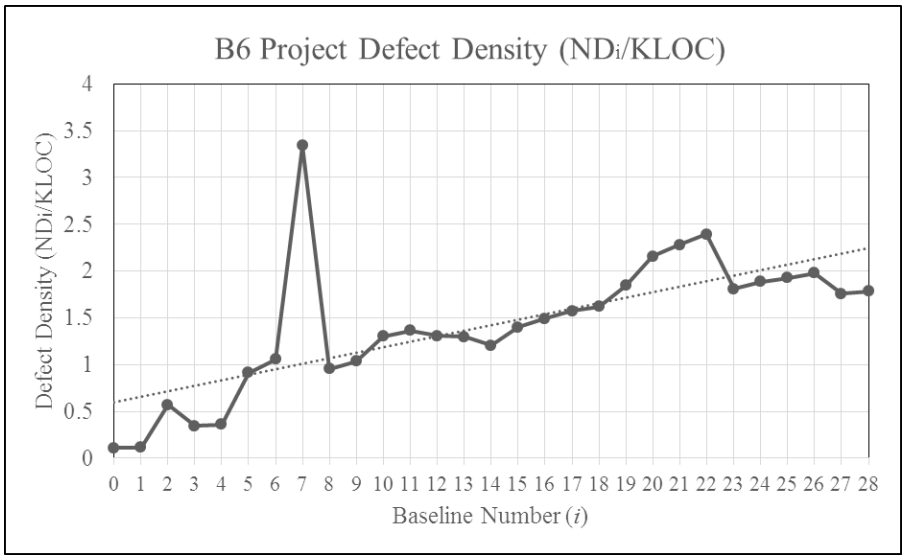

Figure 25. Defect density of the B6 Project

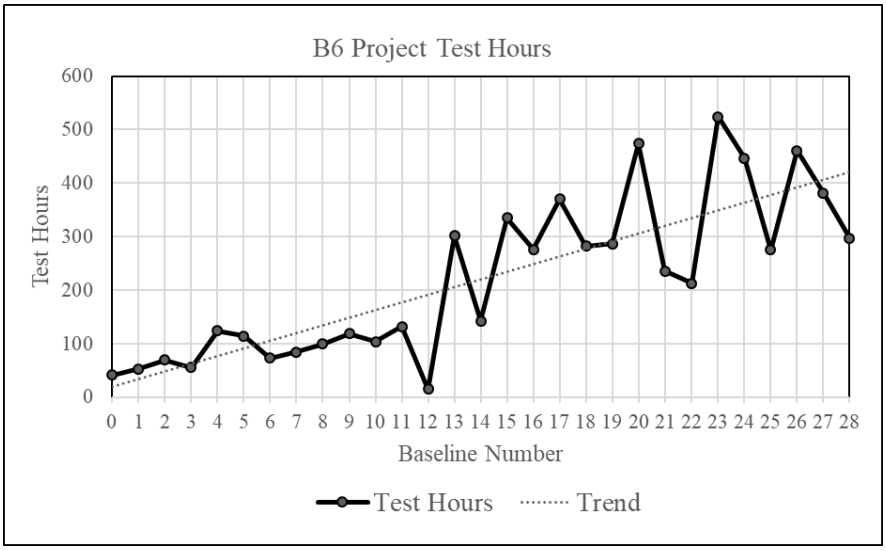

Figure 26. Test Hours of the B6 Project

As shown in Figure 25, the *NDi/KLOC* spikes dramatically on release 7. This indicates an increasing number of defects due to a slight increase in test hours from releases 6 to 7 as shown in Figure 26. Furthermore, the defect density has decreased significantly on release 8 as shown in Figure 25, indicating that testers may have focused their test effort on open defects and may have closed some defects. As testers continue to test the software, the defect density gradually increases. The defect density is trending upward as testers

increase the test hours toward the end of the development cycle and find more defects. Table B-5 of Appendix B shows that the project defect density averaged 1.78 defects per KLOC, which is within the industry standard limit of 3 [160].

3. Organizational Complexity Measure

The B6 Project plan lists nine organizational units, and their responsibilities are identical to those in the B4 Project, as shown in Table A-1 of Appendix A.

Figure B-1 of Appendix B identifies the nodes, links, frequencies, and levels of importance of the communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit during the 28-month development phase. Figures B-2, B-3, and B-4 of Appendix B show the context diagrams of the rest of the nodes, links, communication frequencies, and levels of importance of the communications between the organizations of the B6 Project. We use Figures B-1, B-2, B-3, and B-4 of Appendix B to generate the N2 chart as shown in Table 50. The N2 chart represents nine organizational units and 40 links in the B6 Project.

Table 50. N2 chart of the B6 Project

	IPT	PMA	Project Team	BFM	Contracting Unit	T/F Unit	DT/OT Unit	Total
IPT		3	3	3	1			10
PMA	5							5
Project Team	3	2						5
BFM	2		2					4
Contracting Unit	3							3
Contractor Unit			2	1	1			4
SDU			2			1	1	4
DT/OT Unit		2						2
T/F Unit			3					3
Total links								40

From Equation (3.3) and Tables B-5 and B-6 of Appendix B, we calculate the index function of each organizational unit that is associated with its links. Table B-7 of Appendix B presents the organizational complexity of each organizational unit and the total organizational complexity of 7.64.

The organizational complexity of the B6 Project is very similar to the B4 Project because the B6 Project is the next block of software of the B4 Project. Each organizational unit has to some degree contributed to the negative effect upon the B6 Project in terms of schedule delay and cost overrun due to the time required to process the frequent exchange of information for decision-making.

As shown in Table B-7 of Appendix B, IPT has the most tasks and the greatest number of communications in the project, which, in turn, have contributed to the highest organizational complexity (*IF* = 2.52) among project organizations. CU has the fewest number of tasks and the lowest number of communications, which, in turn, have contributed to the lowest organizational complexity (*IF* = 0.2) among project organizations.

4. Geographical Distribution of Teams Measure

From the B6 Project plan, we identify 14 sites situated in 4 cities and across 2 time zones. Using Equation (3.4), the degree of geographical distribution of teams (*GD*) is determined by adding up the number of sites (*s*), locations (*l)*, and time zones (*t*) as shown in Table B-8 of Appendix B. In this case, we compute *GD* and obtain 20 (14 + 4 + 2). This *GD* is very close to the *GD* of the B4 Project as determined by Table A-11 of Appendix A. Similar to the B4 Project, this high *GD* value suggests that collaborative works between teams in the B6 Project should be infrequent, which may contribute to a higher risk of schedule delay or budget shortfall. However, these organizations in the B6 Project adjusted their project schedules and held weekly status meetings for overcoming the barrier of geographical distribution of teams to achieve the project deliverables.

5. Requirements Volatility Measure

Using Equation (3.5) and the B6 Project profile shown in Table 49, we calculate the value of requirements volatility in the B6 Project. The overall requirements volatility measure (*RVM*) of the B6 Project is as follows:

$$RVM = \frac{CR}{BR} = \frac{2{,}725}{2{,}089} = 1.3$$

where *CR* denotes the number of changes in requirements (new requirements, modifications of requirements, and deletions of requirements) during the 28-month period, and *BR* denotes the baseline of the software release, which is measured by the number of requirements.

Since the project started in February of 2017, data for the number of requirement changes were available for this measurement. We obtain the data of requirements baseline 0 through baseline 26 from the B6 Project requirements metric report and present them in Table B-9 of Appendix B.

As shown in Figure 27, we plot the requirements volatility against the baseline version number with baseline 0 being the start of this new project. The requirements volatility (CR_i / BR) is high in baselines 4, 7, 12, 15, 21, and 25, indicating either uncertainty of the software system or poor understanding of the environment and system. Uncertainty generally is caused by indecisive project planning decisions. The requirements volatility (CR_i / BR) eases after baseline 26, indicating that the number of requirement changes decreases. Furthermore, as indicated in Figure 27, requirement changes tend to decrease toward the end of the development life cycle.

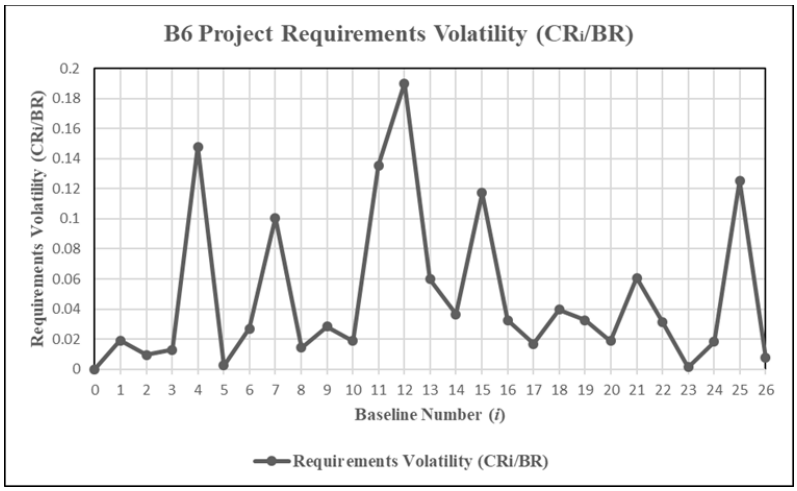

Figure 27. Requirements volatility for the B6 Project

6. Number of Different Job Position Types

The B6 Project plan lists 44 job position types that are identical to those in the B4 Project, as shown in Table A-13 of Appendix A. The B6 Project plan also describes the responsibilities of the 44 job position types that are the same as those in the B4 Project, as shown in Table A-14 of Appendix A. Using Equation (3.6), we calculate *JPT* by counting all job position types and obtain the value of 44.

A high value of *JPT* indicates that the B6 Project has varieties of tasks and task priorities that can affect the overall project performance. People who worked in these 44 types of job positions interact, exchange, and process information in many different ways that contribute to project complexity. Some of the people in these job positions support several projects, and their work schedules are based on the number of hours allocated for supporting that projects.

7. B6 Project Complexity Profile

Using the PCMM, we have calculated the complexity values of the B6 Project. Table 51 shows the complexity profile of the B6 Project.

Table 51. B6 Project complexity profile

Complexity Measure	PCMM Value	Results from previous works and conclusions drawn based on the IPT-related literature
Number of personnel required for the development effort (people)	35.08 people per year	36 people per year based on projects with an annual budget of $5.3 million [151]–[153]
Defect density (defects per KLOC)	1.78	≤ 3 [160]
Organizational Complexity (*IF*)	7.64	5.49 based on Schwandt's study [158]
Geographical distribution of teams (*GD*)	20	Reasonable when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]
Requirements volatility (*RVM*)	1.3	< 1 [162]
Number of different job position types (*JPT)*	44	Sensible and consistent when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]

Regarding the measurement of the number of personnel required for the development effort, the PCMM estimated that 35.08 people are needed for the B6 Project. However, the project plan listed 34 people. This indicates a moderate risk related to the size of the team because the project team has 34 people and may lack the personnel resources as estimated by the PCMM. This analysis shows that the PCMM is fairly consistent and accurate in estimating the level of complexity of the project in terms of the number of people required for the annual budget compared to the B6 Project plan.

Because the development phase was completed in the B6 Project, the test data were available to compute the defect density. When the B6 Project started in February of 2017, the project manager used the defect density value of the B4 Project, which is 0.299 defects per KLOC, and estimated the defect density for the B6 Project as 0.299 defects per KLOC because the B4 Project was the previous block of software of the B6 Project as illustrated in Figure 19. However, the PCMM estimated 1.78 defects per KLOC based on the empirical data. This indicates low risk related to the stability of the released software because the defect density measurement is less than 3 defects per KLOC as specified by the software industry standard [160]. As expected, the defect density of the B6 Project should be higher than the defect density of the B4 Project because additional functions are added to the B6 Project and the number of change requests in the B6 Project is much higher than the number of change requests in the B4 Project. From this perspective, the PCMM is reasonable and consistent in estimating the level of complexity based on the test data of the B6 Project.

In regard to the organizational complexity measurement, the PCMM estimated a value of 7.64 for the organizational complexity. This is consistent with the results of the B4 Project as both projects have identical organizations and similar patterns of communications.

The PCMM estimated a value of 20 for the measurement of the geographical distribution of teams. This measurement is also consistent with the B4 Project because both projects have similar geographical distribution of teams. This measurement indicates medium risk related to the degrees of collaboration between teams because the Allen curve [119] is still applicable. Nevertheless, the PCMM is consistent with a typical Navy software acquisition program because the IPT and project teams adapt to local norms and they have established protocols for communication according to specific purposes for internal and external customers [197].

Because the development phase was completed in this case, the requirements traceability report was available to compute the requirements volatility. When the B6 Project started in February of 2017, the project manager used the requirements volatility value of the B4 Project, which is 0.388, and estimated the requirements volatility for the

B6 Project as 0.388. Nevertheless, the PCMM estimated the requirements volatility measurement as 1.3 based on the empirical data. This indicates high risk related to requirements creep and the understanding of the software requirements as well as use cases because the *RVM* is greater than 1 as indicated in the literature [162]. As expected, the *RVM* of the B6 Project should be higher than the *RVM* of the B4 Project because additional requirements are added to the B6 Project. From this view, the PCMM is consistent and reasonable in estimating the level of complexity based on the requirements traceability report of the B6 Project.

In regard to the measurement of the number of different job position types, the PCMM estimated 44 different job position types based on the B6 Project plan. This is expected because both projects B4 and B6 have identical types of job positions. From this perspective, the PCMM is consistent in estimating the level of complexity based on the B6 Project plan.

The overall complexity level of the B6 Project is calculated as the average of the complexity scores of six measures as shown in Table B-16 of Appendix B. As illustrated in Table B-16 of Appendix B, the overall complexity level of the B6 Project is 4, which is moderate complexity.

Regarding the risk level of the B6 Project, we use Table A-22 of Appendix A to score the six metrics in Tables B-10, B-11, B-12, B-13, B-14, and B-15. The overall risk level of the B6 Project is computed as the average of the six risk scores of the six measures in Table B-17 of Appendix B. We rate the risk level of the project based on degree of impact in terms of cost, schedule, and performance on the project, as shown in Table A-23. As presented in Table B-17 of Appendix B, the overall risk level of the B6 Project is 2, which is medium risk.

Table 52 shows the complexity level and risk level of each metric in the B6 Project.

Table 52. The complexity level and risk level of the B6 Project

Metric	Project Complexity Score	Complexity Level	Risk Level
1. Number of personnel required for the development effort	5	High	High
2. Defect density	3	Low	Low
3. Organizational Complexity	4	Moderate	Medium
4. Geographical distribution of teams	4	Moderate	Medium
5. Requirements volatility	5	High	High
6. Number of different job position types	3	Low	Low
Overall risk level of the B6 Project	4	Moderate	Medium

In sum, the PCMM determines that the B6 Project has moderate complexity and medium risk. The results of the PCMM provide insights on four areas of possible concerns related to the B6 Project.

First, the PCMM determines that the measure of the number of personnel required for the development effort is high risk, which means that the number of people assigned to the project may be insufficient and the project schedule may potentially slip of more than four weeks. Second, the PCMM determines that the measure of organizational complexity is medium risk, which indicates the frequent exchange of information for decision-making and the high number of tasks and communications required for the project. This may cause a cost increase of 10% to 20% to the budget of the project. Third, the PCMM determines that the measure of geographical distribution of teams is medium risk, which may be a

concern for a potential schedule slip of 2 to 4 weeks. Four, the PCMM determines that the measure of requirements volatility is high risk, which indicates a high number of change requests and a potential of a project cost increase of more than 20%.

The PCMM assessment of the B6 Project demonstrated the consistency of the PCMM because the B6 Project was a moderate-risk project according to the B6 Project manager and the IPT senior management review. Nevertheless, the project manager was in control of the project and was able to mitigate the risk of insufficient personnel resources and the risk of requirements creep. The project manager delivered the software on schedule as well as within budget.

C. B8 WINDOWS-BASED DATABASE SOFTWARE PROGRAM

Similar to the B6 Project, the purpose of the B8 Project is to develop a Windows database application for fleet users to use as a tool for mission planning and data analysis. In fact, the B8 Project is the next software block of the B6 Project, and it uses a hybrid development method that is between an agile method and waterfall development method in which the project delivers the software in increments. The B8 software program not only fixes some software defects reported in the B6 software program but it also provides new capabilities by implementing additional requirements requested from the fleet. The B8 Project is broken up into separate program increments which may are representative of the different software builds. Each software build completes a subset of requirements and contains phases that are similar to those of the B6 Project. Figure 28 shows an overview of the B8 Project, which includes project teams, management teams, the B8 Project plan, and SRS as well as databases and elements of both hardware and software. In addition, the project has IT systems supporting ad hoc audits of new requirements, project reviews, and configuration management. Test teams and organizations such as DT, OT, trainers, and fleet representatives are responsible for testing the system.

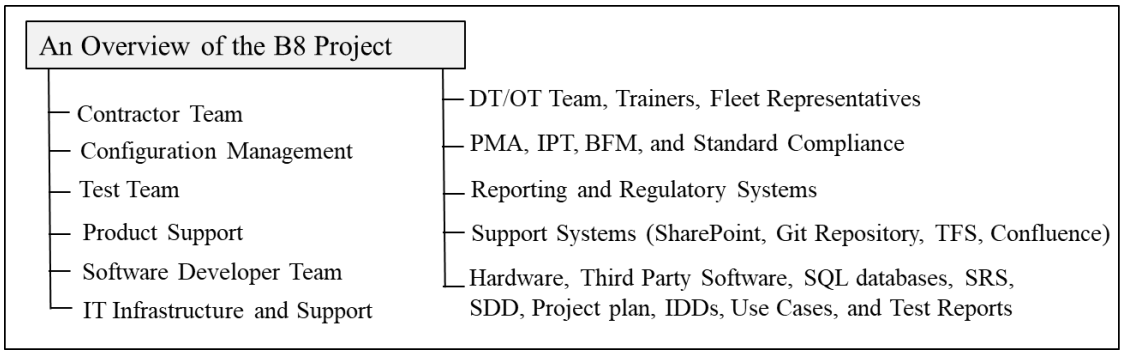

Figure 28. A context diagram of the B8 Project

The B8 Project plan identifies 23 software work products and 25 deliverables that include project documents. It lists the cost of the project is $4.3 million. The B8 Project's organizational chart shows 28 team members. Table C-1 of Appendix C shows some stakeholders' roles and functions listed in the project plan. In addition, Table C-2 of Appendix C presents the PMA, IPT, and PT that are involved in milestone reviews and status reviews during the first 20-month development phase.

The B8 Project plan indicates that the integrated development environment is Microsoft Windows 10 and the C# object-oriented programming language. The IDD documents and test procedures list two external interfaces. Software developers determined and noted in the IDD documents that two external interfaces were easy to implement because these interfaces have few classes that require few methods to query and retrieve data or to perform simple computations for analyzing the data. The B8 test plan lists 43 test procedures that link to 43 use cases. Since the project started in October of 2019, data related to change requests were available during the first 20-month development phase. It listed 1,029 change requests that included 533 new change requests, 32 modifications, and 464 deletions during a 20-month period. The project started with a baseline of 2,822 requirements. At the end of the 20-month period, it had 2,891 requirements.

The number of requirements in the B8 Project is slightly more than the number of requirements in the B6 Project. However, the development phase of the B8 Project is also shorter than the development phase of the B6 Project because the funding of the B8 Project is less than the funding of the B6 Project. After reviewing both project plans B4 and B6, the B8 Project manager interviews the project managers of the B4 and B6 Project. In the

end, the B8 Project manager decides to follow the same approach as the B6 Project and concludes that 80% of requirements will take less than 1 week to implement while 20% of requirements will take between 2 to 4 weeks. Hence, the B8 Project manager determines that 2,313 requirements (80% of requirements) will be easy to implement and the remaining 20% requirements (578 requirements) will be nominal to implement.

Since the B8 Project is the next software block of the B6 Project, both projects have the same IPT management team. Both projects share the software expertise among the team members. Hence, the B8 Project plan indicated that requirements understanding of the software development team was high and the technical risk was nominal. In addition, the process capability of the project team was nominal.

Software developers noted in the SDD that the project has one algorithm that was easy (simple data manipulation and timing not an issue) to develop. Developers determined and noted in the use case documents that they have developed 43 use cases. They noted that 38 use cases were easy to generate, and 5 use cases required a nominal amount of time to create the operational scenarios.

With the information from the project plan, IDD, use case documents, SDD, test plan, test procedures, build reports, and RTR, we create the B8 Project profile as shown in Table 53.

Table 53. B8 Project profile

Project Property	Value	Source of Information
Estimated Yearly Budget	$4.3 million	Project plan
Team Size	28	Project plan
Duration of the Development Phase	20 months	Project plan
Baseline Requirement (BR)	2,822 requirements at project start	RTR
Number of requirements at the end of the project	2,891 requirements	RTR

Project Property	Value	Source of Information
Number of Change Requests (CRs)	1,029 CRs (533 new, 32 modifications, and 464 deletions)	RTR
Number of “difficult” requirements	289 requirements (10%)	Allocating 10% of requirements as “difficult” based on the interview of the B6 Project manager and reviewing the B6 Project plan
Number of “nominal” requirements	289 requirements (10%)	Allocating 10% of requirements as “nominal” based on the interview of the B6 Project manager and reviewing the B6 Project plan
Number of “easy” requirements	2,313 requirements (80%)	Applying the 80–20 rule that is based on the interview of the B6 Project manager and the review of the B6 Project plan
Understanding of requirements [24]	high	Project plan
Technical Risk [24]	nominal	Project plan
Process Capability [24]	nominal	Project plan (CMMI [191] level 3)
Number of External Interfaces	2 (“easy”)	IDD and test procedures
Number of Critical Algorithms [24]	1 (“easy”)	Software design document (SDD) and project plan
Number of use cases	43 (38”easy” and 5 “nominal”)	Use case documents and test plan
Number of lines of code (LOC)	449,836	Build report (SLOC count)

With the project profile shown in Table 53, we apply the PCMM and estimate the B8 Project complexity in the next section.

1. Measure of Number of Personnel Required for the B8 Project

We use the data from the B8 Project profile shown in Table 53 to compute and obtain a software size value of 2,227.7, as shown in Table C-3 of Appendix C. In addition, we use the COSYSMO weight factors of cost drivers shown in Table 17 of Chapter III to calculate and obtain the effort weight factor of 0.77, as shown in Table C-4 of Appendix C.

Finally, we use Equation (3.1) to calculate the number of person-month required (*PM*) [24] in the B8 Project and obtain 27.87 people as shown in Table C-5 of Appendix C. Adding the project reserves of 10% to cover residual risks in the project [174], we determine that the B8 Project needs annual funding for 30.66 people (27.87 times 1.1) or $4.474 million (30.66 times $145,932). In this case, the number of personnel required for the development effort estimated by the PCMM is reasonable and not exceeding 10% of the projected number of personnel required as specified in the B8 Project plan.

2. Defect Density Measure

Since the project started in October of 2019, we have test data to measure defect density. Table C-6 of Appendix C shows the B8 Project monthly test report, taken from the empirical project data. Figures 29 and 30 present the plots of Table C-6 of Appendix C.

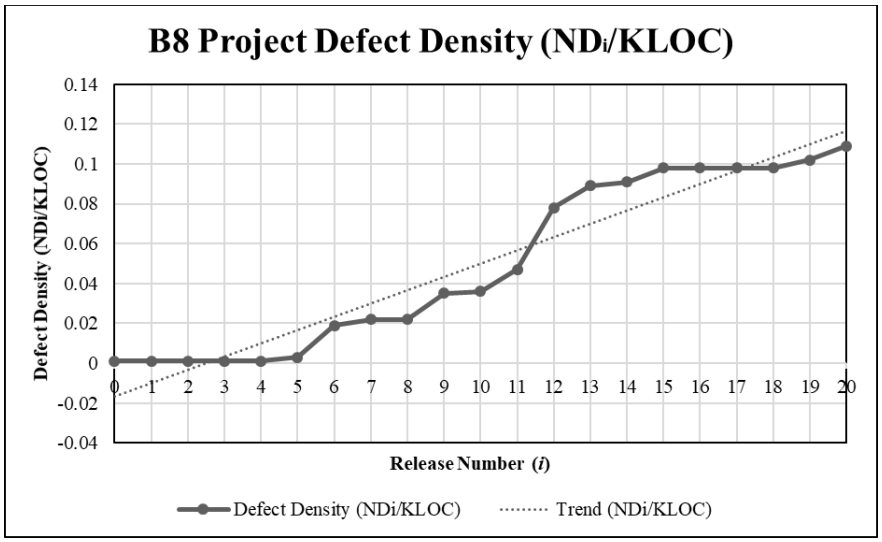

Figure 29. Defect density of the B8 Project

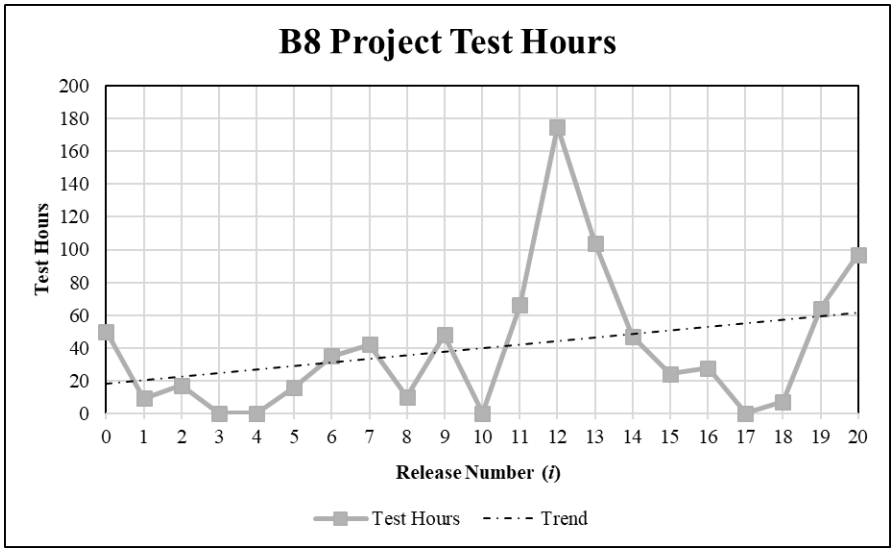

Figure 30. Test hours of the B8 Project

As shown in Figure 29, the defect density trend line is upward toward the end of the development phase. The trend line of test hours is also upward as illustrated in Figure 30. The defect density (*NDi/KLOC*) increases substantially from releases 5 to 6, from 8 to 9, and from 11 to 12 when the test hour increases at the same time. This indicates that testers may be focused on testing while also writing test procedures. The defect density (ND/KLOC) continues trending upward as testers increase the test hours toward the end of

the development cycle. Table C-6 of Appendix C shows that the project defect density averaged 0.109 defects per KLOC, which is within the industry standard limit of 3 [160].

The defect density of the B8 Project showed in Figure 29 fluctuates less than that of the defect density of the B6 Project showed in Figure 25, indicating that the B8 software program is more stable than the B6 software program. In addition, the defect density of the B8 Project fluctuates more than that of the B4 Project showed in Figure 21, indicating that the B8 software program is less stable than the B4 software program. A stable software program has a better project performance than a non-table software program in terms of cost and schedule. Hence, project managers should use the defect density metric as an indicator of project performance.

3. Organizational Complexity Measure

The B8 Project plan listed nine organizational units, and their responsibilities are identical to those in the B4 Project, as shown in Table A-1 of Appendix A.

Figure C-1 of Appendix C shows the nodes, links, frequencies, and the level of importance of communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit during the 20-month period. Figures C-2, C-3, and C-4 of Appendix C show the context diagrams of the rest of the nodes, links, communication frequencies and importance values between the organizations of the B8 Project. We use Figures C-1, C-2, C-3, and C-4 to generate the N2 chart as shown in Table 54. The N2 chart represents nine organizational units and 40 links in the B8 Project. All nine organizational units communicate and exchange information to perform project tasks.

Table 54. N2 chart of the B8 Project

	IPT	PMA	Project Team	BFM	Contracting Unit	T/F Unit	DT/OT Unit	Total
IPT		3	3	3	1			10
PMA	5							5
Project Team	3	2						5
BFM	2		2					4
Contracting Unit	3							3

	IPT	PMA	Project Team	BFM	Contracting Unit	T/F Unit	DT/OT Unit	Total
Contractor Unit			2	1	1			4
SDU			2			1	1	4
DT/OT Unit		2						2
T/F Unit			3					3
Total links								40

From Equation (3.3) and Tables C-7 and C-8 of Appendix C, we calculate the index function of each organizational unit that is associated with its links. Table C-9 of Appendix C presents the organizational complexity of each organizational unit and the total organizational complexity of 7.88.

The organizational complexity of the B8 Project is very similar to the organizational complexity of the B6 Project because the B8 Project is the next block of software of the B6 Project. Each organizational unit has to some degree contributed to the negative effect upon the B8 Project in terms of schedule delay and cost overrun due to the time required to process the frequent exchange of information for decision-making.

As shown in Table C-9 of Appendix C, IPT has the greatest number of tasks and the highest number of communications in the project that contribute to the highest organizational complexity (*IF* = 2.52) among project organizations. CU has the fewest number of tasks and the lowest number of communications that contribute to the lowest organizational complexity (*IF* = 0.2) among project organizations.

4. Geographical Distribution of Teams Measure

From the B8 Project plan, we identified 14 sites situated in 4 cities and across 2 time zones. Using Equation (3.4), the degree of geographical distribution of teams (*GD*) is determined by adding up the number of sites (*s*), locations (*l)*, and time zones (*t*) as shown in Table C-10 of Appendix C. In this case, we compute *GD* and obtain 20 (14 + 4 + 2). This *GD* is identical to the *GD* in the B6 Project. This high *GD* value suggests that collaborative works between teams in the B8 Project should be infrequent, which may contribute to a higher risk of schedule delay or budget shortfall.

Similar to the B6 Project, these organizations in the B8 Project adjusted their project schedules and held weekly status meetings for overcoming the barrier of geographical distribution of teams to achieve the project deliverables.

5. Requirements Volatility Measure

Using Equation (3.5) and the B8 Project profile shown in Table 53, we calculated the value of requirements volatility in the B8 Project. The overall requirements volatility measure (*RVM*) of the B8 Project is as follows:

$$RVM = \frac{CR}{BR} = \frac{1{,}029}{2{,}822} = 0.36$$

where *CR* denotes the number of changes in requirements (new requirements, modifications of requirements, and deletions of requirements) during the 20-month period, and *BR* denotes the baseline of the software release, which is measured by the number of requirements.

Because the project started in October of 2019, we obtain the data of requirements baseline 0 through baseline 20 from the B8 Project requirements metric report and present them in Table C-11 of Appendix C.

As shown in Figure 31, we plot the requirements volatility against the baseline version number with baseline 0 being the start of this new project. The requirements volatility (CR_i / *BR*) is high in baselines 4, 11, 14, and 19 indicating either uncertainty of the software system or poor understanding of the environment and system. Uncertainty generally is caused by indecisive project planning decisions. The requirements volatility (CR_i / *BR*) eases after baseline 20, indicating that the number of requirement changes decreases. Furthermore, as indicated in Figure 31, requirement changes tend to decrease toward the end of the development life cycle.

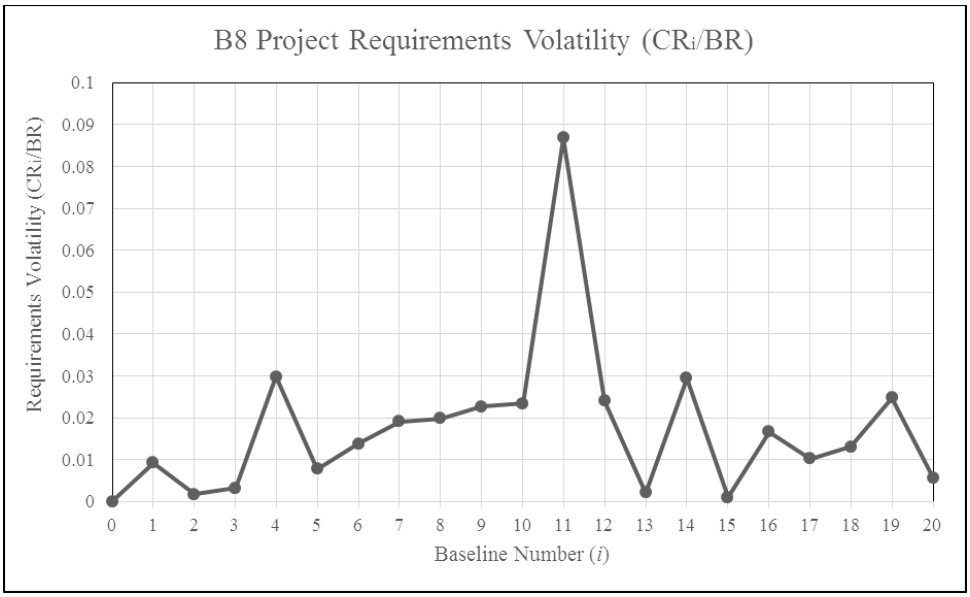

Figure 31. Requirements volatility of the B8 Project

6. Number of Different Job Position Types Measure

The B8 Project plan lists 44 job position types that are identical to those in the B4 Project, as shown in Table A-13 of Appendix A. The B8 Project plan also describes the responsibilities of the 44 job position types that are the same as those in the B4 Project, as shown in Table A-14 of Appendix A. Using Equation (3.6), we calculate *JPT* by counting all job position types and obtain the value of 44.

This high *JPT* value suggests that the B8 Project have varieties of tasks and task priorities that can affect the overall project performance. Similar to the B4 Project and the B6 Project, people who worked in these 44 types of job positions interact, exchange, and process information in many different ways that contribute to project complexity. Some of the people in these job positions support several projects, and their work schedules are based on the number of hours allocated for supporting that projects.

7. B8 Project Complexity Profile

Using the PCMM, we have calculated the complexity values of the B8 Project. Table 55 shows the complexity profile of the B8 Project.

Table 55. B8 Project complexity profile

Complexity Measure	PCMM Value	Results from previous works and conclusions drawn based on the IPT-related literature
Number of personnel required for the development effort (people)	30.66 people per year	29.46 people based on projects with an annual budget of $4.3 million [151]–[153]
Defect density (defects per KLOC)	0.109	≤ 3 [160]
Organizational Complexity (*IF*)	7.88	5.49 based on Schwandt's study [158]
Geographical distribution of teams (*GD*)	20	Reasonable when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]
Requirements volatility (*RVM*)	0.36	< 1 [162]
Number of different job position types (*JPT)*	44	Sensible and consistent when compared to a typical Navy software acquisition program with an annual budget of less than $5 million [197]

Regarding the measurement of the number of personnel required for the development effort, the PCMM estimated that 30.66 people are needed for the B8 Project. But based on the estimated annual project cost shown in the project plan, we could fund 29.46 people ($4.3 million / $145,932) yearly for the B8 Project. However, the project plan listed 28 people per year and allocated some funding for project reserves. This indicates a moderately low risk related to the size of the team because the 28 people on the B8 Project is less than 30.66 people estimated by the PCMM. This analysis shows that the PCMM is

fairly consistent and accurate in estimating the level of complexity in terms of the number of people required for the annual budget compared to the B8 Project plan.

Because the development phase was completed in the B8 Project, the test data were available to compute the defect density. When the B8 Project started in October of 2019, the project manager used the defect density value of the B6 Project, which is 1.78 defects per KLOC, and estimated the defect density of the B8 Project as 1.78 defects per KLOC because the B6 Project was the previous block of software of the B8 Project as shown in Figure 19. Nevertheless, the PCMM estimated 0.109 defects per KLOC based on the empirical data. This indicates low risk related to the stability of the released software because the defect density measurement is less than 3 defects per KLOC as specified by the software industry standard [160]. As expected, the defect density of the B8 Project should be lower than the defect density of the B6 Project because the number of change requests in the B8 Project is much less than the number of change requests in the B6 Project. From this perspective, the PCMM is reasonable and consistent in estimating the level of complexity based on the test data of the B8 Project.

In regard to the organizational complexity measurement, the PCMM estimated a value of 7.88 for the organizational complexity. This is consistent with the results of the B6 Project as both projects have identical organizations and similar patterns of communications.

The PCMM estimated a value of 20 for the measurement of the geographical distribution of teams. This measurement is also consistent with the B6 Project because both projects have the same geographical distribution of teams. This measurement indicates medium risk related to the degrees of collaboration between teams because the Allen curve [119] is still applicable. Nevertheless, the PCMM is consistent with a typical Navy software acquisition program because the IPT and project teams adapt to local norms and they have established protocols for communication according to specific purposes for internal and external customers [197].

Because the development phase was completed in this case, the requirements traceability report was available to compute the requirements volatility. When the B8

Project started in October of 2019, the project manager used the requirements volatility value of the B6 Project, which is 1.3, and estimated the requirements volatility for the B6 Project as 1.3. Nevertheless, the PCMM estimated the requirements volatility measurement as 0.36 based on the empirical data. This indicates low risk related to requirements creep and the understanding of the software requirements as well as use cases because the *RVM* is less than 1 as indicated in the literature [162]. As expected, the *RVM* of the B8 Project should be lower than the *RVM* of the B6 Project because the number of change requests in the B8 Project is less than the number of change requests in the B6 Project. From this view, the PCMM is consistent and reasonable in estimating the level of complexity based on the requirements traceability report of the B8 Project.

In regard to the measurement of the number of different job position types, the PCMM estimated 44 different job position types based on the B8 Project plan. This is expected because both projects B6 and B8 have identical types of job positions. From this perspective, the PCMM is consistent in estimating the level of complexity in terms of number of different job position types based on the B8 Project plan.

The overall complexity level of the B8 Project is calculated as the average of the six complexity scores of the six measures in the PCMM as shown in Table C-18 of Appendix C. As illustrated in Table C-18 of Appendix C, the overall complexity level of the B8 Project is 3 (low complexity).

Regarding the risk level of the B8 Project, we use Table A-22 of Appendix A to score the six metrics in Tables C-12, C-13, C-14, C-15, C-16, and C-17. The risk level of the project is defined in Table A-23, which is based on degree of impact in terms of cost, schedule, and performance on the project. The risk level of the B8 Project is computed as the average of the six complexity scores of the six measures shown in Table C-19 of Appendix C. As presented in Table C-19 of Appendix C, the overall risk level of the B8 Project is 1.5 (low risk).

Table 56 shows the complexity level and risk level of each metric in the B8 Project.

Table 56. The complexity level and risk level of the B8 Project

Metric	Complexity Score	Complexity Level	Risk Level
1. Number of personnel required for the development effort	4	Moderate	Medium
2. Defect density	1	Simple	Low
3. Organizational Complexity	4	Moderate	Medium
4. Geographical distribution of teams	4	Moderate	Medium
5. Requirements volatility	2	Complicated	Low
6. Number of different job position types	3	Low	Low
Overall complexity level and risk level of the B8 Project	3	Low	Low

In sum, based on the assessment of the PCMM, the B8 Project has a low complexity and the risk level of the project is low. The PCMM provides insights on three areas of possible concerns related to the B8 Project.

First, the number of personnel assigned to the project (28 people per year listed in the project plan) is very close to the number of personnel estimated by the PCMM (27.87 people per year). However, because we need to allocate project reserves of 10% to cover residual risks, this may create a risk in the schedule. Thus, the PCMM reflects that concern as a medium risk, which means a potential schedule slip of 2 to 4 weeks.

Second, the organizational complexity may be a concern because how the team engages in communications and how the team makes decision affect the project execution.

Hence, the PCMM indicates that concern as a medium risk, which means a potential cost increase of 10% to 20% to the project budget.

Third, the geographical distribution of teams may be a concern as indicated by the PCMM because the project needs some collaborative work between teams and a rich and robust communications platform for executing the project. The PCMM shows this concern as a medium risk, which means a potential schedule slip of 2 to 4 weeks.

The PCMM assessments of the B8 Project demonstrated the consistency of the PCMM because the B8 Project was a low-risk project according to the IPT senior management review. The project manager monitored the team progress and kept the budget and schedule under control. The project manager mitigated the risk of insufficient personnel resources. According to the IPT senior management review, the B8 Project delivered the software on schedule and within budget.

8. Comparison of Complexity Profile of Three Engineering Projects

A comparison of complexity profile of the three engineering projects (the B4 Project, B6 Project, and B8 Project) appears in Table 57. As expected, the PCMM indicates that the B6 Project has a higher complexity level and risk level than the B8 Project and the B4 Project. This result is consistent with the project data and the IPT senior management review. Table 57 indicates that the B6 Project is worse than that of the B4 and B8 Projects to deliver the product on time and within budget. Note that the B6 Project costs more than 20% of the B4 and B8 Projects. The B6 Project schedule is 4 to 8 months longer than the B4 and B8 Projects. These observations are consistent with the moderate-complexity rating and the medium-risk rating by the PCMM.

Because the B4 Project was completed, data were available to measure defect density and requirements volatility of the B4 Project. On the other hand, both B6 and B8 Projects are still continuing. Therefore, the project managers of the B6 and B8 Projects use earlier periods of test data and measures the defect density and requirements volatility.

In all three cases, the PCMM is consistently estimated the complexity levels and the risk levels of the projects. As shown in Table 57, the B6 Project is the most complex

project and has the highest risk level among the three projects. Thus, the B6 Project has the highest requirements volatility and the highest defect density among the three projects. Furthermore, Table 57 indicates that the B6 Project requires more personnel for the development effort than the B8 Project and the B4 Project. In addition, the B6 Project has the longest period of development and the highest number of defects per KLOC among the three projects.

All three engineering projects have similar levels of organizational complexity. In general, if interdependence is high among organizational units, then the time, cost, and effort necessary to coordinate the process will be high. All three engineering projects have similar values in the measure of different job position types and in the measure of geographical distribution of teams. This makes sense because the B8 Project is the continuation of the B6 Project, and the B6 Project is the continuation of the B4 Project. Note that a long project duration generally increases the risk of schedule slippage and cost overrun as demonstrated by the PCMM in the B6 Project complexity profile shown in Table 57.

Table 57. A comparison of complexity profile of three engineering projects

Complexity Measure	**B4 Project** (24 months)	**B6 Project** (28 months)	**B8 Project** (20 months)
1. Number of personnel required for the development effort (people/year)	17.09	35.08	30.66
2. Defect density (defects per KLOC)	0.299	1.78	0.109
3. Organizational Complexity	7.66	7.64	7.65
4. Geographical distribution of teams	19	20	20
5. Requirements volatility	0.388	1.3	0.36
6. Number of different job position types	44	44	44
Complexity Score	2.67	4	3

Complexity Measure	B4 Project (24 months)	B6 Project (28 months)	B8 Project (20 months)
Risk Score	1.167	2	1.5
Complexity Level	Low	Moderate	Low
Risk Level	Low	Medium	Low

The range of values of the measure of number of personnel required for the development effort is between 17 people per year to 35 people per year. These values are typical for a Navy software development project where the annual budget of the project is between 3.3 million to 5.3 million.

In regard to the measure of defect density, the range of values of this measure is between 0.1 defects per KLOC to 0.32 defects per KLOC in a typical Navy software development project where the software program lines of code is between 450 KLOC to 870 KLOC. Note that the B6 Project has 1.78 defects per KLOC, which is not typical due to the high number of defects in the software program.

Regarding the measure of organizational complexity, the range of values of this measure is between the values of 5.4 and 7.7 in a typical Navy software development project where the project normally involves 9 to 10 organizations in the IPT.

The range of values of the measure of requirements volatility in a typical Navy software development project is between the values of 19 and 20 where the government agencies and the contractors collaborate onsite and offsite to development the software program.

The range of values of the measure of geographical distribution of teams in a typical Navy software development project is between the values of 19 and 20 where the government agencies and the contractors collaborate onsite and offsite to development the software program.

In regard to the measure of the number of different job position types, a typical Navy software development project has a range of values from 44 to 46 types of job positions where the project normally involves 9 to 10 organizations in the IPT with an annual budget between 3.3 million to 5.3 million.

The program managers of the B4, B6, and B8 Projects define the program success according to the following project outcomes:

- The software has no priority 1 or 2 defects. A Priority 1 defect is defined as a critical and "must fix" defect because it prevents the accomplishment of an operational capability. A Priority 2 defect is defined as a very important and no work-around defect but also "need to fix" defect because it adversely affects technical, cost, or schedule risks to the project.
- IPT and senior management have reviewed the project metrics that include the number of defects and priorities, defect density, the number of statement of requirements achieved, and man-hours spent on the project.
- The software meets all statements of requirements.
- The PMA receives a letter of certification of the software from the program manager to enter the operational test (OT).
- The OT unit completes the OT test and sends the OT report to the PMA, recommending releasing the software to the fleet.

The B4, B6, and B8 Projects all achieved the desired project outcomes. Based on the B6 Project outcomes stated in test reports, statements of requirements, and certification letters as well as project expenditures drawn from financial data such as labor, contracts obligation, and materials, the B6 Project shows a greater number of test hours, defects, change requests, and the number of personnel required for the development effort than the B4 and B8 Projects. The program manager and the PMA conclude that the B6 Project is more complex and has a higher level of risk than the B4 and the B8 Projects.

In short, as demonstrated by the three Navy software projects, the PCMM and associated methods to measure complexity can help systems engineers and program managers analyze project complexity and reason through their decisions related to systems engineering. The PCMM can help systems engineers perform the following items:

- To estimate the number of personnel required for the development effort of the SoI,
- To understand project complexity and risk in terms of requirements stability and product quality control via defect density measurement during or after the completion of the project,
- To assess the impacts from a given complexity value of the number of locations of the team on team communications for collaborative work, and
- To assess the impacts of a given complexity value of the number of different job positions on the integration of technical processes.

9. Sensitivity Analysis

In Section C of Chapter III, we performed a sensitivity analysis on both the number of personnel required for the development effort and the requirements volatility of the PCMM. In this section, we use the project data from the three engineering projects and perform a sensitivity analysis on the measure of defect density because this measure has a major effect on the complexity level and risk level of the project. The number of test hours and the number of defects in each use case are obtained from Table A-7 of Appendix A, Table B-5 of Appendix B, and Table C-6 of Appendix C. As shown in Table 58, the number of defects per test hours increases by 112.82% (0.22 / 0.195) between Project B4 and Project B6, the defect density of the B6 Project also increases by 595.31% (1.78 / 0.299). The sensitivity value of this measurement is 5.27 (595.31% / 112.82%).

As illustrated in Table 58, the number of defects per test hours decreases by 379.31% (0.22/0.058) between Project B6 and Project B8, the defect density of the B8 Project also decreases by 1,633% (1.78/0.109). The sensitivity value of this measurement

is 4.3 (1,633% / 379.31%). Hence, the sensitivity value of the PCMM is stable, dropping from 5.27 to 4.3.

Table 58. Sensitivity analysis of the defect density measurement in the three engineering projects

Project Data	B4 Project	B6 Project	B8 Project
Defect density (defects / KLOC)	0.299	1.78	0.109
Total number of test hours	1,339.5	6,384.85	838.5
Total number of defects	261	1,408	49
Defects / Test hours	0.195	0.22	0.058

Since there is little change in the measurements of organizational complexity, geographical distribution of teams, and the number of different job position types among the B4, B6, and B8 Projects, we will not perform a sensitivity analysis on these measures.

In sum, we performed a sensitivity analysis on the defect density of the PCMM. The analysis shows that in regard to the defect density measurement, the PCMM is sensitive to the changes in the technical parameters of the project. At the same time, the PCMM is stable within the ranges of the technical parameters that specify in the project profile. Therefore, the PCMM does sustain the different levels of complexity.

10. Analysis of Three Main PCMM Measures Related to Project Cost, Schedule, and Performance

Because we have three metrics (number of personnel required for the development effort, defect density, and requirements volatility) in the PCMM that affect project complexity to the greatest extent in our three engineering projects, let's analyze each metric and draw some conclusions related to project cost, schedule, and performance.

Since the B6 Project is the most complex and has a higher risk than both the B4 and B8 Projects, our project correlation analysis of the three main PCMM measurements will be focused on the B6 Project, as shown in Tables 59, 60, and 61. According to the project managers and the IPT senior management review, all three projects met the requirements, budgets, performances, and schedules for the deliverables.

Based on the results of three engineering projects, we conclude that the PCMM's measure of the number of personnel required for the development effort correlates to project cost and may also be correlated to project schedule and performance.

Table 59. Analysis of project cost and the measure of the number of personnel required for the development effort

	B4 Project	**B6 Project**	**B8 Project**
Project cost per year	$3.3 million	$5.3 million	$4.3 million
Project duration (months)	24	28	20
Number of requirements	1,994	2,822	2,891
Number of change requests	803	2,725	1,029
KLOC	870.559	790.487	449.836
PCMM measure: number of personnel required for the development effort (people/year)	17.09	35.08	30.66
Measure correlates to project cost.	The B6 Project requires more personnel for the development effort and costs more than the B4 and B8 Projects.		
Measure may be correlated to project schedule.	The B6 Project has a longer project duration and a greater number of personnel required for the development effort than the B4 and B8 Projects		
Measure may be correlated to performance.	Both the B6 and B8 Projects require more personnel for the development effort and have more requirements than the B4 Project.		

In Table 60, we conclude that the PCMM's measure of defect density correlates to performance and may also be corrected to project cost and project schedule.

Table 60. Analysis of project performance and the measure of defect density

	B4 Project	**B6 Project**	**B8 Project**
Project cost per year	$3.3 million	$5.3 million	$4.3 million
Project duration (months)	24	28	20
Number of requirements at the end of the project	1,994	2,822	2,891
Number of change requests	803	2,725	1,029
KLOC	870.559	790.487	449.836
Number of test hours	1,339.5	6,384.85	838.5
Number of defects	261	1,408	49
Defects / test hours	0.195	0.22	0.058
PCMM measure: defect density	0.299	1.78	0.109
Measure may be correlated to project cost.	The B6 Project has higher defect density and costs more than the B4 and B8 Projects.		
Measure may be correlated to project schedule.	The B6 Project has higher defect density and a longer project duration than the B4 and B8 Projects.		
Measure correlates to performance.	The B6 Projects has higher defect density and has more change requests than the B4 and B8 Project.		

In Table 61, we conclude that the PCMM's measure of requirements volatility correlates to project cost and may also be correlated to project schedule and performance.

Table 61. Analysis of project cost and the measure of requirements volatility

	B4 Project	**B6 Project**	**B8 Project**
Project cost per year	$3.3 million	$5.3 million	$4.3 million
Project duration (months)	24	28	20
Number of requirements at the end of the project	1,994	2,822	2,891
Number of change requests	803	2,725	1,029
KLOC	870.559	790.487	449.836
Number of test hours	1,339.5	6,384.85	838.5
Number of defects	261	1,408	49
Defects / test hours	0.195	0.22	0.058
PCMM measure: requirements volatility	0.388	1.3	0.36
Measure correlates to project cost.	The B6 Project has higher requirements volatility and costs more than the B4 and B8 Projects.		
Measure may be correlated to project schedule.	The B6 Project has higher requirements volatility and a longer project duration than both the B4 and B8 Projects.		
Measure may be correlated to performance.	The B6 Projects has higher defects per test hours and has more change requests than the B4 and B8 Project.		

The overall correlation analysis of the three main PCMM's measures are shown in Table 62. As shown in Table 62, the correlation between the measure of the number of personnel required for the development effort and the project cost is positive. This indicates that as the number of personnel required for the development effort increases, the project cost also increases, and vice versa.

On the other hand, the correlation between the measure of defect density and the project performance is negative, as shown in Table 62. This suggests that as the defect density value increases, the project performance decreases, and vice versa.

The correlation between the measure of requirements volatility and the project schedule is positive, as illustrated in Table 62. This suggests that as the requirements volatility value increases, the project duration is also longer, and vice versa.

Table 62. Correlation Analysis of the three main PCMM's measures

PCMM Measure	Project Attribute	Correlation
1. Number of personnel required for the development effort	Cost	positive
	Schedule	negative
	Performance	negative
2. Defect density	Cost	positive
	Schedule	positive
	Performance	negative
3. Requirements volatility	Cost	positive
	Schedule	positive
	Performance	negative

11. Analysis of the PCMM Measures Related to the Risk Levels of the Project

The overall correlation analysis of the PCMM's measures and the risk levels of the project are shown in Table 63. As shown in Table 63, the correlation between the measure of number of personnel required for the development effort and the risk levels of the project is positive. This indicates that as the number of personnel required for the development effort increases, the risk levels of the project also increase, and vice versa.

Similarly, the correlation between the measure of defect density and the risk levels of the project is also positive, as shown in Table 63. This suggests that as the defect density value increases, the risk levels of the project also increase, and vice versa.

The correlation between the measure of requirements volatility and the risk levels of the project is positive, as illustrated in Table 63. This suggests that as the requirements volatility value increases, the risk levels of the project also increase, and vice versa.

The rest of the PCMM measures also has a positive correlation to the risk levels of the project.

Table 63. Correlation Analysis of the PCMM's measures and risk level of the project

PCMM Measure	Risk Levels of the project	Correlation
1. Number of personnel required for the development effort	Low, medium, and high	positive
2. Defect density	Low, medium, and high	positive
3. Organizational Complexity	Low, medium, and high	positive
4. Geographical distribution of teams	Low, medium, and high	positive
5. Requirements volatility	Low, medium, and high	positive
6. Number of different job position types	Low, medium, and high	positive

12. Complexity Reduction and Mitigation for the B6 Project

Reduction of project complexity helps project managers simplify their project structures and communications platform. Reduction in any of these areas creates opportunities for cost saving while strengthening the execution of project tasks and increasing the focus on customers. Project managers use complexity reduction as a management tool to streamline business processes and optimize information systems with project objectives.

In regard to reduction of the number of personnel required for the development effort of the B6 Project, the project manager can reduce the number of system requirements, major interfaces, critical algorithms, and use cases in each block by dividing the number of system requirements into several increments of software blocks. In addition, the software lead can consolidate multiple functional requirements into a common platform or framework to improve design efficiency.

The project manager can reduce the cost driver EM_i of the project by performing the following: (1) increase the level of requirements understanding to very high through peer reviews and technical interchange meetings, (2) increase the process capability to very high via a robust communications platform, and (3) reduce the technical risk of the project to very low through unit tests and by reducing interdependency in designs and processes.

In regard to reduce defect density of the B6 Project, the project manager and the software lead can reduce the number of defects (ND_i) in each baseline ($KLOC$) by performing the following: (1) conduct unit tests and code walk-through procedures that developers should abide by, (2) set coding standards, (3) develop loosely coupled code and make the code easier to maintain, (4) standardize on one IT platform that provides configuration controls of code releases, (4) leverage on a modular design of code and reuse that code to improve design efficiency, (5) limit the amount of code change within a software unit, and (6) prototype code in the early stages of development.

In terms of reducing organizational complexity in the B6 Project, the project manager can do the following: (1) reduce the number of tasks among organizational units by focusing on tasks that are critical to customers' needs, (2) reduce the frequency of the

information exchange among organizational units by setting a priority and a baseline in project reporting, and (3) clarify organizational roles and decision-making processes to best serve the project teams while also streamline on business processes and information systems.

In regard to reduce geographical distribution of teams, the project manager can decrease the number of locations (*l*), time zones (*t*), and sites (*s*) of each organizational unit in the project by performing the following: (1) reduce the number of project activities that take place in multiple locations, and (2) change the problem definition or approach in such possible ways that reduces complexity of project activities.

In regard to reduce requirements volatility of the B6 Project, the project manager can lower the number of change requirements in each baseline of the release by performing the following: (1) keep project tasks and requests separate, (2) implement a formal change approval process, (3) keep a change log, (4) use online collaboration tools, and (5) perform requirements prototyping, and (6) conduct peer reviews.

In addition, the program manager can freeze the project scope early in the development cycle and postpone change requests that are not immediately needed by the customers, thus, raising the hurdles for new capability requirements and other expansion activities that add complexity costs. The program manager may also need to limit the addition of new requirements late in the development life cycle.

To reduce the number of different job position types in the B6 Project, the project manager can cut down the number of job position types (pi) in each organizational unit by performing the following: (1) combine and eliminate some reporting tasks and review activities via process improvement, and (2) increase the competency level and control maturity of a project.

In sum, the PCMM prescribes a systematic way to calculate the complexity value of any engineering project. The model is consistent and predictable in that it relates project effort to personnel resources in order to comprehend design effort, to perform system integration, prototyping, and testing. In addition, the complexity profile identifies the

complexity metrics and communicates the significance of those metrics to stakeholders, aiding in decision-making.

Identifying issues early in a project's life cycle can be challenging because complexity increases and covers interactions between components. The complexity values of the B6 Project obtained from the PCMM can help systems engineers and project managers pinpoint and then analyze what aspects of the B6 Project are potential risks of schedule slippage and then study them.

In general, it is more expensive to correct an error discovered late in the project life cycle. Systems engineers can use a complexity value computed from the PCMM as a guide for project reviews and comparisons of alternatives. Complexity is not always bad, but sometimes even desirable. In fact, in some cases, maintaining some degree of complexity is essential to effective operations and keen risk management. For example, in a high-tech hardware manufacturer, the manufacturer has redundant supply sources to prevent periodic supply disruptions. The manufacturer mitigates the risk of supply disruptions by adding complexity to the supply chain. As a result, the manufacturer has steady manufacturing operations even though the operational cost of the business has increased. On the other hand, complexity reduction helps many organizations simplify their business strategy, organizational structures, processes, deployment of products, and information technology. By strengthening the execution of project tasks and the relationships with the customers, organizations can potentially reduce their cost of doing businesses. In short, organizations use complexity reduction to optimize business processes and streamline information systems with project objectives.

D. CHAPTER SUMMARY

In sum, this chapter has addressed the four questions in the beginning of this chapter. First, the complexity values from the PCMM are consistent with the results from the software industry standard [160], [162], previous works found in the related literature [28], [157], [158], and the IPT-related literature [197]. With the exception of the value of requirements volatility in the B6 Project, the complexity values of all six measures of the

PCMM in the three cases are within the acceptable limits of software industry standard, previous works found in the related literature, and the IPT-related literature.

Second, the PCMM accurately describes the complexity properties of real-world systems that are similar to the three Navy acquisitions projects that were analyzed in these project. The applicability of this model to a broader range of projects requires further study.

Third, the PCMM can potentially work as a prescription for complexity analysis in real-world systems by providing a complexity model and associated methods to measure and analyze complexity in three Navy software development projects. The results from three software projects are consistent with the PCMM. Project parameters and characteristics of the three projects are adequate and representative of a typical Navy software project based on results from previous works, International Project Management Association (IPMA) experts' opinions identified in the literature review, and conclusions drawn from the IPT-related literature.

The rationale and analyses of the three engineering projects based on the PCMM are reasonable. The project profile for each use case is based on project plan, SRS, IDD, test plan, requirements metric reports, and test reports at the beginning of the planning and development phase of the acquisition life cycle. The project profile includes project cost, project duration, the number of requirements, the number of change requests, and other key elements used by the PCMM. However, the applicability of this model to a broader range of projects requires further study.

Fourth, the PCMM can potentially help systems engineers and project managers to identify, understand, and assess the impacts a given complexity value on real-world systems through comparisons of the complexity measurements of three engineering projects.

The model is consistent and predictable in that it relates project effort to personnel resources in order to comprehend design effort, to perform system integration, prototyping, and testing. In addition, the complexity profile identifies the complexity metrics and communicates the significance of those metrics to stakeholders, aiding in decision-making. Practitioners often use previous or similar projects to compare complexity measurements

and then confidently perform approved measures by the project manager. Nevertheless, the applicability of this model to a broader range of projects requires further study.

Overall, in the three projects, the measure of the number of personnel required for the development effort, defect density, and requirements volatility are the three major measures that most contribute to complexity in the projects. These three measures are correlated to project cost and performance. From the complexity profile of the B6 Project, we have shown that limiting numerosity and interdependence results in decreasing organizational complexity. Furthermore, reducing the number of change requests related to the requirements of the SoI decreases requirements volatility. In addition, the patterns of requirement changes over time and the number of test hours conducted during the development phase have substantial influence on defect density. Thus, systems engineers should aim to reduce complexity by limiting numerosity and interdependence in a system or a project.

In sum, the PCMM is workable because:

- The results from three software projects are consistent with the PCMM.
- Project parameters and characteristics of the three engineering projects are adequate and representative of a typical Navy software project based on results from previous works, International Project Management Association (IPMA) experts' opinions identified in the literature review, and conclusions drawn from the IPT-related literature.
- Practitioners often use previous or similar projects to compare complexity measurements and then confidently perform approved measures by the project manager.

We can use the PCMM as a guide to assess complexity in the design and management of complex projects. Practitioners will ultimately use previous products or projects of a similar size to compare complexity measurements to gain insight on what works for their project and then to confidently perform approved measures by the project manager.

As noted in Chapter I, complexity can occur in systems (computer hardware, software, network infrastructures, etc.), projects (process, constraints, regulations, etc.), and environments (interfacing systems, cultural variety, social span, etc.). The three projects serve as test beds that demonstrate the validity of the PCMM for computing complexity metrics.

The PCMM would serve as a guide to assess project complexity. The results of our work on the PCMM and on these three software projects show that as a project increases in complexity, more interactions and dynamics among the project participants and stakeholders make it more difficult for a project to meet all requirements and schedules. However, we can reduce complexity with the better structures, standard tools, procedures, and techniques as well as with familiarity of the SoI.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION AND FUTURE WORK

> Systems engineers need measures of complexity if they are going to work on complex systems.
>
> —Robert C. Harney [29],
> Professor of Systems Engineering, Naval Postgraduate School

Chapter V starts with a summary of the main outcomes of this research and then follows with a review of the research's contributions relative to the current literature. Next, we address some shortcomings of the research method and results. In the following section, we suggest some areas of potential future research to extend this work. Finally, we propose some ideas for future endeavors inspired by this research.

A. SUMMARY

The intent of this dissertation is to provide engineers and project managers with a model and a systematic approach that identifies key complexity measures in engineered systems for project management and risk assessment, as well as potential approaches to complexity reduction. To achieve these goals, this research developed a unique project complexity measurement model (PCMM) and associated methods to measure complexity in a project. The PCMM, which consists of a set of complexity measures, creates a complexity profile of the SoI that is relevant and useful for project risk assessment and program complexity assessment. In addition, the PCMM and associated methods to measure complexity can help systems engineers and program managers reason through their decisions related to systems engineering.

A review of the relevant literature reveals that there is a gap in the current SE body of knowledge concerning the topics of system affordability and SE advanced measurements [178]. In the topics of system affordability and advancements of SE measurements, there is a need of a use-case study to link between complexity to the cost and schedule of complex systems engineering projects [178]. To study the relationship of

this link, we need to identify a model of complexity and measurement methods to analyze project complexity and to provide insight for improvement of system affordability.

Furthermore, with the increasing focus on the value and cost of the system across the acquisition life cycle, models for complexity analysis, analysis of system affordability, and techniques for reducing system complexity are becoming more important in the DOD acquisition arena. There exists a gap between what is available in the DOD systems acquisition programs today and the complexity measurement models that could be used in the early phases of systems engineering on those military acquisition products [129], [130]. For example, many DoD programs are underestimating the difficulty of software development efforts. Collecting and analyzing complexity metrics in the early phases of development can reduce the risk of budget overruns and schedule slippage due to system development difficulty.

The PCMM and associated methods to measure complexity in a project have extended the current SE body of knowledge by bringing together into one holistic model that consists of several different metrics proposed by several different researchers. This model includes measures of number of personnel required for the development effort, requirements volatility, defect density, geographical distribution of project teams, organizational complexity, and the number of different types of job positions. This approach incorporates principles of risk management and project management in software engineering by relating them to risk analysis and decision-making to support high quality systems engineering.

In addition, this research has covered a use case of SE body of knowledge in the topic of SE advanced measurement where systems engineers and project managers use the PCMM and associated methods to measure complexity and assess project risks to ensure affordability. For example, the PCMM and associated methods link project complexity to systems development cost, schedule, performance, and risk. As a result, the PCMM provides project managers and IPT management teams with common practices in the areas related to the review of work products, roles and responsibility expectations, accountability demands, and affordability.

The main contributions of this research are as follows:

- A systematic approach to assess project complexity and explore design alternatives early in the process as well as potential approaches to complexity reduction,
- A model for identifying key complexity measures in engineering projects and relationships between different elements of complexity within systems for project management,
- A quantitative model and associated methods for expressing and communicating systems and project complexity issues to project managers, systems engineers, manufacturers, and stakeholders, and
- Advancement in the state of practice in the project risk assessment of Navy software systems.

In Chapter IV, the results of three projects (the B4, B6, and B8 Projects) demonstrated that the measurements of the number of personnel required for the development effort, defect density, and requirements volatility are the three major measures that contribute to complexity in the Navy software projects. The results of our work on these three projects demonstrated that as a project increases in complexity, more interactions and dynamics among the project participants and stakeholders make it more difficult for the project to meet all requirements and performance expectations.

This dissertation discussed ways for reducing project complexity while strengthening the execution of project tasks. As mentioned in Chapter IV, we can reduce project complexity with better structures, standard tools, procedures, and techniques as well as with familiarity of the SoI. For example, in the B6 Project, the program manager and software design lead need to identify software architectures that best address a client-server system (i.e., the logical 3-tier system with a data layer, an application layer, and a presentation layer). The software design lead needs to train software developers on software design patterns and the C# coding standards. The systems engineers need to identify and address SoS engineering concerns by conducting trade studies, modeling and prototyping of the SoI, and communicating with software developers to clarify the system's requirements. Software developers need to determine the correct design by adapting proven designs and following the C# coding standards.

The PCMM correlated the complexity attributes of the project to management effort, design effort, and testing. The three projects showed that three of the six complexity measures were significant for computing project complexity (number of personnel required for the development effort, requirements volatility, and defect density in terms of number of defects per KLOC). Although not all six measures were significant, the three projects demonstrated that the most complex project has higher values in these three measures than the less complex projects. In addition, the most complex project has a higher risk of failure in regard to the budget, schedule, and performance. However, the applicability of this model to a broader range of projects requires further study.

In sum, the three projects serve as test beds that demonstrate partial validity of the PCMM for computing complexity metrics. In addition, the complexity profile from each use case identifies the six complexity measures (number of personnel required for the development effort, defect density, organizational complexity, geographical distribution of project teams, requirements volatility, and the number of different job position types) and communicates the significance of those measures to stakeholders, aiding in decision-making.

B. CONCLUSIONS

Systems engineers and project managers must identify interdependencies of core processes and tasks and manage them to minimize project risk. Systems engineers and project managers should apply resources not only in project oversight and clarifying project requirements but also in developing prototypes and conducting simulations of integration processes to minimize project risk.

This research demonstrated that the PCMM and associated methods to measure complexity provide project managers practical measures to estimate an engineering project's complexity, allowing them to analyze project complexity and reason through their decisions related to systems engineering early in the planning and design process. Systems engineers and project managers can use this research to determine the level of complexity in a project or a system, and from that assessment, plan development efforts and apply resources in appropriate places. In placing value on a system's degree of complexity,

system engineers and project managers can predict a specific range of possible impacts. With that evaluation, targeted actions can mitigate these impacts.

The likely impact of a given complexity value on engineering systems can be deleterious. A high value returned by the complexity function might imply the necessity of additional time and effort in development, integration, testing, and maintenance. Complexity is an indicator of project risk and, in general, we should aim to keep a complexity value low. The results of the software projects in Chapter IV support this observation. In addition, a high complexity value assigned to a system might predict difficulty in conducting exhaustive system testing due to the complex operational environment and/or a very large system.

System design, system requirements, changes of technology, team dynamics, organizational policies and procedures, and external stakeholders are factors that contribute to complexity. A reasonable thought is that complexity changes with time because engineering projects and their environments evolve over time.

The identification of complexity in engineered systems and the PCMM's method of estimating the degree of complexity in any engineering project shed more light on the approach of measuring project complexity by identifying metrics that describe project complexity based on key risk factors in an engineering project. In addition, the PCMM lays the groundwork for complexity measurement standards for engineering projects by applying a set of measures of complexity to the project that are relevant and important to the program managers and systems engineers.

There are several shortcomings of the research method and results. First, the PCMM only provides metrics that are associated with complexity. It does not provide measurement of causal relationships between each complexity factor. For instance, the measure of the number of personnel required for the development effort is related to the project size (i.e., the number of requirements, interfaces, algorithms, and operational scenarios). Defect density measures the number of defects per KLOC. Both measures are related to the project size. In general, given a fixed schedule, a large project requires more people to complete than a small project. A large software project usually contributes to

higher defect density than a small software project because it simply entails more code to write. Thus, there is a causal relationship between the measure of the number of personnel required for the development effort and the measure of defect density.

Second, the PCMM does not provide measurement of recursive relationships between some project attributes and complexity. For instance, the PCMM does not measure the relationship between the number of project work packages and complexity or the relationship between employing emerging technology of the project and complexity. Moreover, a recursive relationship exists between the system development cycle and complexity. For example, a software development cycle typically consists of several phases like creating requirements, designing software, writing code, integration testing, deploying software, and maintaining the code [22], [100]. Software complexity, which can be gauged by the number of lines of code, the length of the procedure, and the number of branches in the code, can affect the maintenance cost of the software [146]. As the LOC increases, "the software becomes more complex and more bugs may be introduced" [146], and, therefore, the maintenance costs also increase [146]. In addition, the software build process can be difficult due to unanticipated dependencies and "complex branching structure" [146]. Large software projects also need substantial "management control" [146] (monitoring and tracking the number of defects, the number of change requirements, and the number of test hours). Hence, a recursive relationship exists between the system development cycle, some project attributes, and complexity.

In sum, this research demonstrated a practical way to measure complexity of engineering projects. Systems engineers and project managers can use this complexity measurement method to provide both relevant information to stakeholders in the form of metrics that contribute to complexity and early guidance concerning actions necessary to accommodate that complexity.

C. AREAS FOR FUTURE RESEARCH

We invested considerable effort in this research to develop a systematic method to estimate the complexity of engineering projects. Opportunities also exist in related research areas that would broaden the SE body of knowledge. Because of partial validation from

the results of the three projects, and not knowing the nature of any of the relationships between complexity factors and complexity, both the pursuit of more empirical data of the project and the performance of regression analysis could lead to better estimates of the relationships between the complexity factors and complexity. In addition, the PCMM and associated methods to measure complexity can be extended to the creation of complexity profiles for small systems and enterprise systems. The PCMM and associated methods can be extended for the following items:

- The complexity in systems of systems, which will be a challenge to measure because of the potential for emergent behaviors, distributed processes and controls as well as conflicting requirements, and
- Other complex entities, such as acquisitions, the software in a system, the system's technical performance, technological maturity across the system, and risk.

Furthermore, the PCMM and associated methods can be applied to assess the effectiveness of approaches that reduce and contain complexity. In addition, the PCMM and associated methods can be applied to assess the recursive relationship between a system development cycle and complexity.

D. FINAL REMARKS

Until this research, systems engineers have used a few project complexity measures only in a specific application. The software projects in Chapter IV demonstrated that the PCMM and associated methods to measure complexity are both practical and useful for engineering projects. This research provides a systematic method to compare the complexity of engineering projects and to determine which project is less risky to manage. Thus, the PCMM and associated methods to measure complexity provide some answers to the research questions presented in Chapter I.

In engineering complex systems, engineers often aim to reduce complexity because causes of complexity are significant and that can affect the system's performance, safety, and reliability of operations. Addressing the causes of complexity is a practical way to reduce project risks in engineering projects. This research provides a tool to measure

project complexity and analyze its effects as well as to interpret CSE activities. Project managers and systems engineers can use this research in heuristics that suggests which kinds of complexity to reduce to improve system affordability. At a minimum, engineers can apply measures of the complexity presented in this research to quantify a design complexity and a development effort. Consequently, systems engineers can design simpler systems or processes according to these measures of complexity.

EPILOGUE

As globalization has grown in recent years, technology has expanded and improved rapidly. Our world is changing very quickly, and we have many complex issues to tackle such as climate change, the COVID-19 pandemic, and flu outbreaks. Complexity science is concerned with systems and problems that are vigorous, unpredictable, and difficult to solve, consisting of many parts and interconnected relationships, and with causes and effects not obviously related. When we talk about complexity, we refer to ideas that are difficult-to-describe, predict, model, and understand. A complex system has multiple interfaces and subsystems as well as multiple levels of interactions.

In this research, we have presented a model and methods that enable engineers to assess and estimate complexity of systems and projects. We have explored some classes of problems that require complex systems knowledge. We have discussed some ideas for how to reduce complexity in a system and project. Finally, we have presented a CSE model that improves our methods for enterprise SE.

This research has demonstrated a step forward in devising a set of measures that provides benefits from having a method to quantify a design complexity and a development effort. Such benefits could help identify areas of possible concern, make projections, and provide inputs to decision analyses such as trade studies.

We are on the edge of a vast and emerging science. One main purpose of this research is to provide a clearer picture of the concept of complexity and an understanding of the complexity that surrounds us. We can ask whether core ideas of the complexity sciences presented in this research can help us address some difficult problems faced by today's society—the spread of disease, the effects of climate change, and the unequal distribution of economic resources. We are making progress in the understanding of complex systems and starting to reveal their secrets—emergence, limited predictability, self-organization, spontaneous order, and evolutionary dynamics. No doubt, we will need more research as we move forward. The underlying journey seems possible, challenging, and exciting.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. COMPLEXITY MEASURES OF THE B4 PROJECT

In this appendix, we show the complexity measures of the B4 Project, which include the number of personnel required for the development effort, defect density, organizational complexity, and geographical distribution of teams as well as requirements volatility and the number of different job position types.

1. Measure of the Number of Personnel Required for the Development Effort

Table A-1 shows the organizational units and their responsibilities in the B4 project.

Table A-1. The B4 Project organizational units and their responsibilities

Organizational Unit	Responsibilities
1. Program Management Activity (PMA)	• Provides funding to programs such as a capability development program (CDP). • Decides programs priorities. • Decides programs hardware and software.
2. Integrated Product Team (IPT)	• Provides lab spaces and network infrastructures to projects. • Decides organizational processes. • Submits CDP. • Manages contracts. • Mitigates program risks.
3. Project Team (PT)	• Manages and executes project requirements. • Participates in critical design review and senior management review. • Provides personnel work status and project metrics to IPT.
4. Business Financial Management (BFM)	• Sends, receives, and processes funding documents. • Provides labor and materials expenditures to the project team.
5. Contractor Unit	• Provides work and services to the project.

Organizational Unit	Responsibilities
6. Software Distribution Unit (SDU)	• Packages and distributes software to trainers, to the DT/OT unit, and to the fleet.
7. Trainer/fleet users Unit (T/F Unit)	• Trains fleet users on how to use fleet software. • Reports software anomalies to the project team.
8. Contracting Unit (CU)	• Provides statements of work and manages contracts.
9. DT/OT Unit	• Performs developmental and operational tests on software.

Table A-2 presents organizational units involved in milestone reviews and status reviews during the 24-month development phase.

Table A-2. Planned reviews from the B4 Project plan

Organizational Unit	Review	Frequency
PMA	• CDP review	One time during the planning phase
IPT	• Senior management review	Quarterly
	• Project management review	Once every two weeks during the planning and development phase
PT	• System and software requirement review • Preliminary design review • Critical design review	Two times during the development phase
	• Technical interchange meeting	Monthly
	• Project milestone review	Three times during the development phase
	• CCB meeting	Once every two weeks

Table A-3 shows the calculation of software size for the B4 Project.

Table A-3. Calculation of software size for the B4 Project

Software size's driver [24]	B4 Project software size
Requirements	(399 nominal requirements)(1) = 399 (1,595 easy requirements)(0.5) = 797.5 Total is 1,196.5 (399 + 797.5)
Interfaces	(3 easy interfaces)(1.7) = 5.1 (1 nominal interface)(4.3) = 4.3 Total is 9.4 (5.1 + 4.3)
Algorithms	(2 easy algorithms)(3.4) = 6.8 (2 nominal algorithms)(6.5) = 13 Total is 19.8 (6.8 + 13)
Operational Scenarios (use cases)	(27 easy use cases)(9.8) = 264.6
Total software size [24] = requirements + interfaces + algorithms + user cases	1,196.5 + 9.4 + 19.8 + 264.6 = 1,490.3

Table A-4 shows the calculation of the B4 Project's effort weight factor.

Table A-4. The B4 Project's effort weight factor

Effort weight factors	Value
Requirements understanding is high [24].	0.77
Technical risk is nominal [24].	1
Process capability is nominal [24].	1
The B4 Project's effort weight factor is calculated [24].	(0.77)(1)(1) = 0.77

Table A-5 shows the calculation of *PM* (person-month) [24] and the number of people required for the development effort. Note that the calibration constant is 0.325, and

the effort weight factor is 0.77 in a typical software development project [24]. As shown in Table A-5, the B4 Project requires 15.54 people to develop the system in 24 months.

Table A-5. The B4 Project development effort

B4 Project development effort	Value
Effort required to build the system = *PM* = (calibration factors)(effort factor)(software size)	(0.325)(0.77)(1,490.3) = 372.95 person-month
B4 Project development effort = (effort required to build the system) / (duration of the development phase)	372.95 / 24 = 15.54 people

2. Defect Density Measure

Table A-6 shows the B4 Project defect density during the 24 months of the development phase. The number of defects for a release (ND_i) represents the cumulative defects unresolved.

Table A-6. B4 Project defect density during the 24 months of the development phase

Release (*i*)	**Month**	**Cumulative defects unresolved (ND_i)**	**KLOC**	**Defect density (ND_i/KLOC)**	**Test Hours**
0	Apr-17	188	910.617	0.206	81
1	May-17	193	926.073	0.208	204
2	Jun-17	200	925.060	0.216	210.5
2	Jul-17	201	925.060	0.217	86
3	Aug-17	201	859.681	0.233	15
3	Sep-17	206	859.681	0.239	85
3	Oct-17	213	859.681	0.247	10
4	Nov-17	213	858.702	0.248	38.5
4	Dec-17	217	858.702	0.252	63
5	Jan-18	218	859.007	0.253	17
6	Feb-18	224	859.081	0.26	157
7	Mar-18	227	859.069	0.264	47
7	Apr-18	231	859.069	0.268	82.5
7	May-18	233	859.069	0.271	5
8	Jun-18	235	859.094	0.273	22
8	Jul-18	238	859.094	0.277	0
9	Aug-18	252	860.018	0.293	83
9	Sep-18	255	860.018	0.296	0
10	Oct-18	134	870.559	0.153	119
10	Nov-18	256	870.559	0.294	14
10	Dec-18	258	870.559	0.296	0
10	Jan-19	259	870.559	0.297	0
10	Feb-19	261	870.559	0.299	0
10	Mar-19	261	870.559	0.299	0

Table A-7 shows the consolidate data of the releases of Table A-6. In Table A-7, the defect density of the B4 Project is 0.299 in the first 24 months (April 2017 to March 2019). The total test hours for that period of 24 months is 1,339.5 hours. 261 defects are reported in the 24-month period.

Table A-7. Consolidated data of the releases of Table A-6

Release (*i*)	Month	Cumulative defects unresolved (ND_i)	KLOC	Defect density (ND_i/KLOC)	Test Hours
0	Apr-17	188	910.617	0.206	81
1	May-17	193	926.073	0.208	204
2	Jul-17	201	925.060	0.217	296.5
3	Oct-17	213	859.681	0.247	110
4	Dec-17	217	858.702	0.252	101.5
5	Jan-18	218	859.007	0.253	17
6	Feb-18	224	859.081	0.26	157
7	May-18	233	859.069	0.271	134.5
8	Jul-18	238	859.094	0.277	22
9	Sep-18	255	860.018	0.296	83
10	Mar-19	261	870.559	0.299	133
Total					1,339.5

3. Organizational Complexity Measure

Figure A-1 shows the context diagram of nodes, links, communication frequencies and levels of importance of the communications between PMA and IPT, between PMA and PT, and also between PMA and the DT/OT unit during the 24 months development phase.

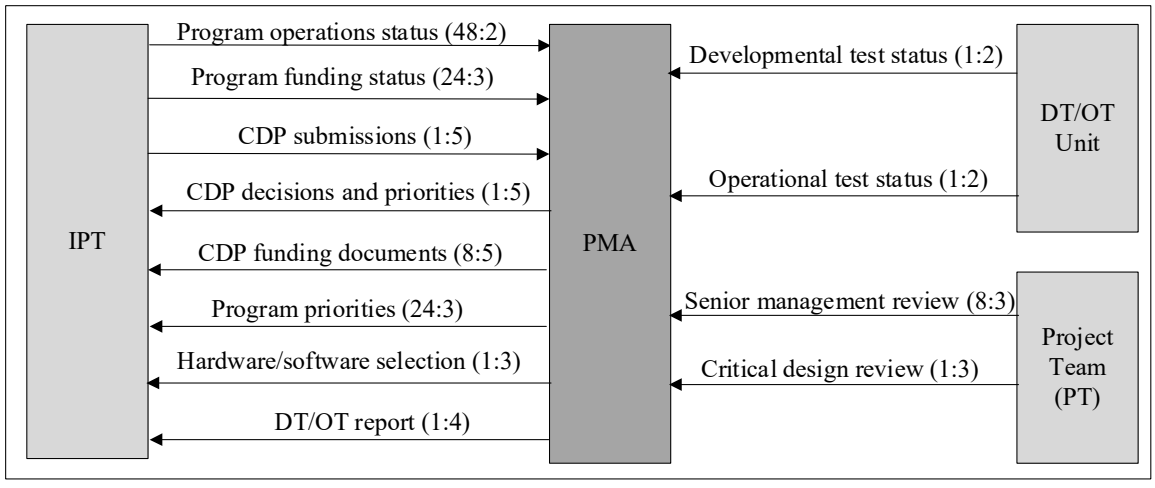

Figure A-1. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit

Figures A-2, A-3, and A-4 show the context diagrams of the rest of the nodes, links, communication frequencies, and importance levels of the communications between the organizations of the B4 Project.

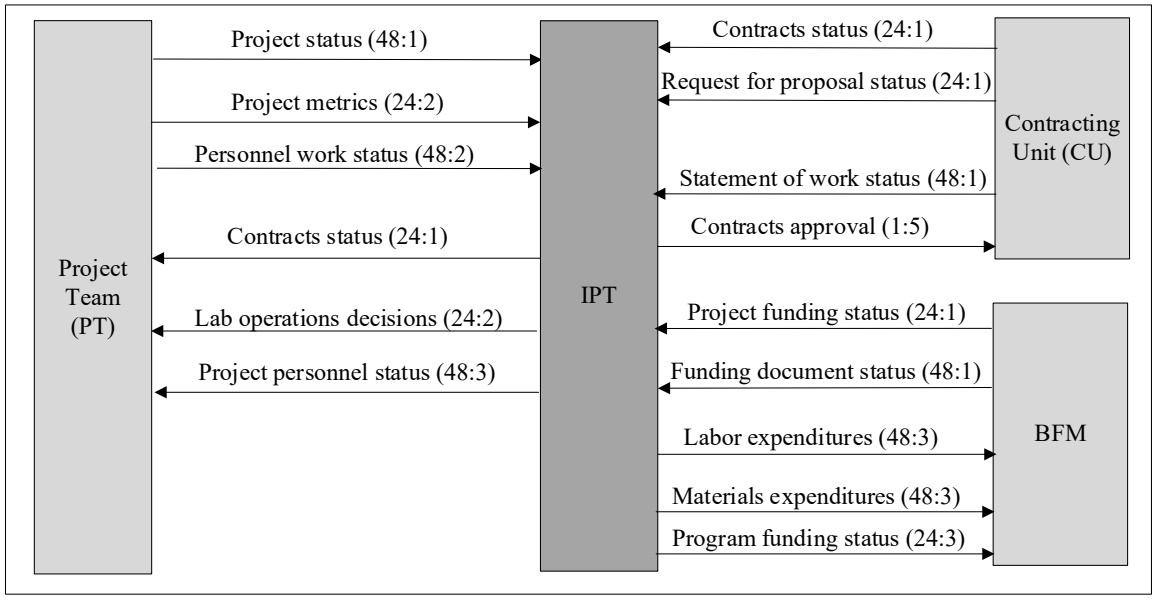

Figure A-2. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between IPT and PT, between IPT and CU, and between IPT and BFM

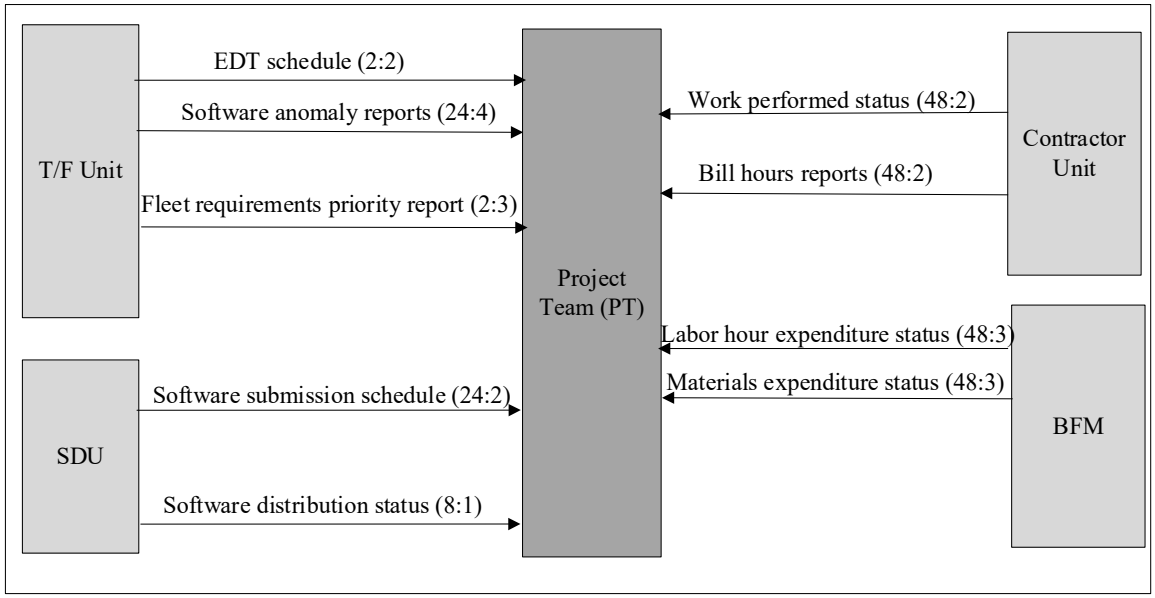

Figure A-3. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PT and the T/F unit, between PT and SDU, between PT and BFM, and between PT and the contractor unit

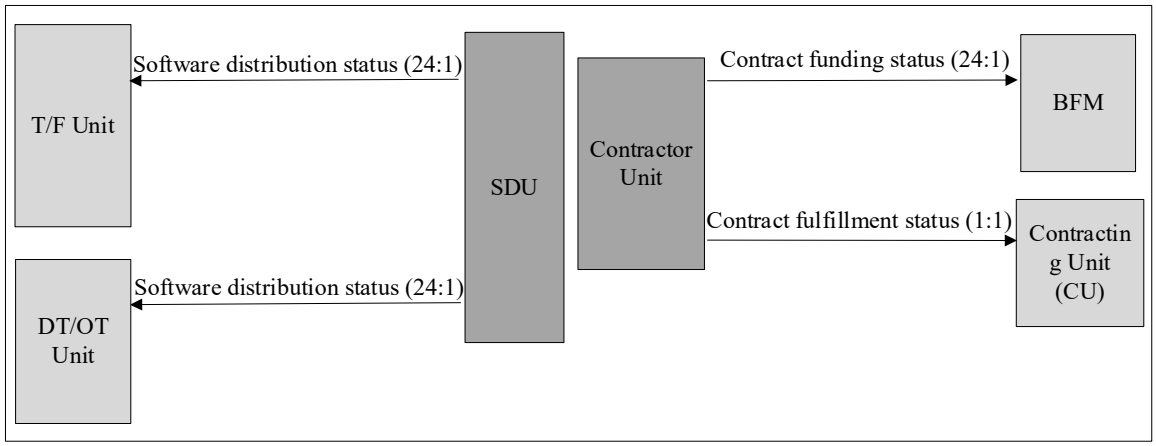

Figure A-4. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between SDU and the T/F unit, between SDU and DT/OT unit, between the contractor unit and BFM, and between the contractor unit and CU

Tables A-8 and A-9 show the frequencies and levels of importance of the communications in the B4 Project during the 24 months of the development phase. These

data are extracted from project status reports, test reports, senior management reviews, and the project plan.

Table A-8. Frequencies of the communications during the entire 24 months of the development phase

	Communication of information	**Frequency**						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
1. IPT	1. Program operations status		48					
	2. Program funding status		24					
	3. CDP submission		1					
	1. Contracts status			24				
	2. Lab operations status			24				
	3. Project personnel status			48				
	1. Labor expenditures				48			
	2. Materials expenditures				48			
	3. Program funding status				24			
	1. Contracts approval decisions					1		
	Subtotal		73	96	120	1		
2. PMA	1. CDP decisions	1						
	2. CDP funding document	8						
	3. Program priorities status	24						
	4. B4 hardware/software selection	1						
	5. B4 DT/OT report	1						
	Subtotal	35						
3. Project Team (PT)	1. Project status	48						
	2. Project metrics	24						
	3. Personnel work status	48						
	1. Critical design review status		1					
	2. Senior management review		8					
	Subtotal	120	9					
4. BFM	1. Project funding status	24						
	2. Funding document status	48						
	1. Labor hour expenditure status			48				

	2. Materials expenditure status			48				
	Subtotal	72		96				
5. Contracting Unit (CU)	1. Contracts status	24						
	2. RFP status	24						
	3. Statement of work status	48						
	Subtotal	96						
6. Contractor Unit	1. Work performed status			48				
	2. Bill hours report			48				
	1. Contract funding status				24			
	1. Contract fulfillment status					1		
	Subtotal			96	24	1		
7. T/F Unit	1. EDT schedule			2				
	2. Software anomaly report			24				
	3. Fleet requirements priority report			2				
	Subtotal			28				
8. DT/OT Unit	1. DT status		1					
	2. OT status		1					
	Subtotal		2					
9. SDU	1. Software submission schedule			24				
	2. Software distribution status			8				
	1. Software release status						24	24
	Subtotal			32			24	24
	Total	322	84	348	144	2	24	24

Table A-9. Levels of importance of the communications during the entire 24 months in the B4 Project

	Communication of information	**Level of importance**						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
1. IPT	1. Program operations status		2					
	2. Program funding status		3					
	3. CDP submission		5					
	1. Contracts status			1				
	2. Lab operations status			2				
	3. Project personnel status			3				
	1. Labor expenditures				3			
	2. Materials expenditures				3			
	3. Program funding status				3			
	1. Contracts approval decisions					5		
2. PMA	1. CDP decisions	5						
	2. CDP funding documents	5						
	3. Program priorities status	3						
	4. Hardware/software decisions	3						
	5. DT/OT report	4						
3. Project Team	1. Project status	1						
	2. Project metrics	2						
	3. Personnel work status	2						
	1. Critical design review status		3					
	2. Senior management review		3					
4. BFM	1. Project funding status	1						
	2. Funding document status	1						
	1. Labor hour expenditure status			3				
	2. Materials expenditure status			3				
5. Contracting Unit (CU)	1. Contracts status	1						
	2. RFP status	1						
	3. Statement of work status	1						
	1. Work performed status			2				
	2. Bill hours report			2				

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
6. Contractor Unit	1. Contract funding status				1			
	1. Contract fulfillment status					1		
7. T/F Unit	1. EDT schedule			2				
	2. Software anomaly reporting			4				
	3. Fleet requirements priority report			3				
8. DT/OT Unit	1. DT status		2					
	2. OT status		2					
9. SDU	1. Software submission schedule			2				
	2. Software distribution status			1			1	1

Table A-10 shows the organizational complexity of each organizational unit. We sum up the contribution of organizational complexity from each of the nine organizational units during the period of 24 months and obtain the total organizational complexity of 7.66.

Table A-10. Organizational complexity of each organizational unit in the B4 Project

Organizational Unit	*IF*				Total
1. IPT interacts with PMA, PT, BFM, and CU.	PMA: [(2)(48)+3(24)+(5)(1)]/[(5)(73)]= 0.474	PT: [(1)(24)+(2)(24)+(3)(48)]/[(5)(96)] = 0.45	BFM: [(3)(48)+(3)(48)+(3)(24)]/[(5)(120)] = 0.6	CU: (1)(5)/[(5)(1)] = 1	2.52
2. PMA interacts with IPT.	IPT: [(5)(1)+(5)(8)+(3)(24)+(3)(1)+(4)(1)]/[(5)(35)] = 0.708				0.708
3. PT interacts with IPT and PMA.	IPT: [(1)(48)+(2)(24)+(2)(48)]/[(5)(120)] = 0.32		PMA: [(3)(1)+(3)(8)]/[(5)(9)] = 0.6		0.92
4. BFM interacts with IPT and PT.	IPT: 134516 [(1)(24)+(1)(48)]/[(5)(72)] = 0.2		PT: [(2)(48)+(2)(48)]/[(5)(96)] = 0.4		0.6
5. CU interacts with IPT.	IPT: [(1)(24)+(1)(24)+(1)(48)]/[(5)(96)] = 0.2				0.2

Organizational Unit	*IF*			Total
6. Contractor Unit interacts with PT, BFM, and CU.	PT: [(2)(48)+(2)(48)]/[(5)(96)] = 0.4	BFM: [(1)(24)]/[(5)(24)] = 0.2	CU: [(1)(1)]/[(5)(1)] = 0.2	0.8
7. T/F Unit interacts with PT.	PT: [(2)(2)+(4)(24)+(3)(2)]/[(5)(28)] = 0.757			0.757
8. DT/OT Unit interacts with PMA.	PMA: [(2)(1)+(2)(1)]/[(5)(2)] = 0.4			0.4
9. SDU interacts with PT, T/F Unit, and DT/OT Unit.	PT: [(2)(24)+(1)(8)]/[(5)(32)] = 0.35	T/F Unit: [(1)(24)]/[(5)(24)] = 0.2	DT/OT Unit: [(1)(24)]/[(5)(24)] = 0.2	0.75
Total Contributions				7.66

4. Geographical Distribution of Teams Measure

Table A-11 shows the geographical distribution of teams from the nine organizational units in the B4 Project. We sum up the number of sites, locations, and time zones. We obtain the value of 19 for the measure of geographical distribution of teams.

Table A-11. Geographical distribution of teams in the B4 Project

	Number of Sites	**Location**	**Time Zone**
1. PMA	1	East-coast city	Eastern time
2. IPT	1	Southwest-1city	Pacific time
3. PT	3	Southwest-1 city	Pacific time
4. BFM	1	Southwest-1 city	Pacific time
5. CU	1	Southwest-1 city	Pacific time
6. Contractor Unit	2	Southwest-1 city	Pacific time
7. T/F Unit	1	Northwest city	Pacific time
8. DT/OT Unit	2	East-coast city and Southwest-2 city	Eastern time and Pacific time
9. SDU	1	Southwest-1 city	Pacific time
Total	13	4	2

5. Requirements Volatility Measure

Table A-12 show the requirements metrics during the 24 months of the development phase. As presented in Table 58, the project starts with a baseline of 2,067 requirements. During a 24-month period, the B4 Project has 803 change requests (356 new + 18 modifications + 429 deletions) and 1,994 requirements.

Table A-12. B4 Project requirements metrics during the 24 months of the development phase

	Requirement (CR_i)			BR = 2,067 requirements	
Baseline	New	Modification	Deletion	Total # of requirements	Requirements volatility = $(CR_i)/(BR)$
0				2,067	0
1	50			2,117	0.024
2	5			2,122	0.002
3	10			2,132	0.004
4	150			2,282	0.072
5	17	1	7	2,292	0.012
6	6			2,298	0.003
7	12	11	2	2,308	0.012
8	4	1	7	2,305	0.005
9			5	2,300	0.002
10	23		311	2,012	0.161
11	7		30	1,989	0.018
12	7		14	1,982	0.010
13	19		8	1,993	0.013
14	20	2	5	2,008	0.013
15	5	2		2,013	0.003
16	5		27	1,991	0.015
17	1			1,992	0
18			1	1,991	0
19	4			1,995	0.002
20	4			1,999	0.002
21	1			2,000	0
22	1			2,001	0
23	4		1	2,004	0.002
24	1	1	11	1,994	0.006
Total	356	18	429		

6. Number of Different Job Position Types

Table A-13 shows 44 types of job position in the B4 Project. Thus, the measure of different job position types (*JPT*) is 44.

Table A-13. Job position types in the B4 Project

Organization Unit	Job Position Type	
1. PMA	a. Program manager b. Executive officer c. Logistics lead d. Business operations manager	
2. IPT	a. IPT site lead b. Military lead c. Office manager d. Chief systems engineer e. Program-related engineering lead	f. Lead technologist g. International programs lead h. Process improvement lead i. Fleet help desk specialist j. Training manager
3. PT	a. Product lead b. Project manager c. Systems engineer d. Design lead e. Requirements lead f. Programmer	g. Software installer h. Test lead i. Configuration manager j. Product documentation lead k. Standards compliance lead l. Tester
4. BFM	a. BFM lead b. Financial analyst c. Financial technician	
5. CU	a. Contracting officer b. Contracting analyst c. Contract specialist	

Organization Unit	Job Position Type
6. Contractor Unit	a. Contract representative b. Site manager c. Technical team lead
7. T/F Unit	a. Fleet liaison b. Trainer c. Fleet representative
8. DT/OT Unit	a. Lead software engineer b. Information assurance specialist c. Technical analyst
9. SDU	a. Program analyst b. Product integrity lead c. Systems engineer lead

Table A-14 shows the responsibilities for the 44 job position types in the B4 Project.

Table A-14. Responsibilities for job position types in the B4 Project

Job Position Type	Responsibility
1. Project sponsor	Define future operational requirements for the fleet. Set program's priorities and allocate funding.
2. Executive officer	Coordinate and decide program's funding and priorities.
3. Logistics lead	Coordinate and distribute hardware to various programs at the PMA level.
4. Business operations manager	Perform supervisory responsibilities associated with the strategic planning, risk management, and administrative and management activities for various programs.
5. IPT site lead	Direct, plan, budget, and manage the execution of programs at the IPT level.

Job Position Type	Responsibility
6. Military lead	Serve as IPT site lead from the military chain of command.
7. Office manager	Provide credit card buys. Maintain training records and create reports for various data calls.
8. Chief systems engineer	Review, evaluate, coordinate, and monitor programs at the IPT level.
9. Program-related engineering lead	Coordinate and allocate funding for program-related engineering and program-related logistics.
10. Lead technologist	Coordinate and execute capability development across multiple product lines and on solution development with other competencies.
11. International programs lead	Collaborate and coordinate with defense science and technology (DST) group and foreign partners to develop new capabilities. Provide support to FMS customers.
12. Process improvement lead	Plan, review, and execute standards compliance and process improvement policies established by the IPT.
13. Fleet help desk specialist	Provide help desk support to fleet users by answering phone calls and responding to emails from the fleet.
14. Training manager	Plan, schedule, coordinate, and conduct process improvement training.
15. Product lead	Direct and lead development of project plans, including project schedule, technical performance, and budget. Approve or disapprove milestone reviews.
16. Project manager	Coordinate, assess, and monitor software development schedules, contracts, budgets, and personnel of the B4 Project.
17. Systems engineer	Perform requirements, functional interfaces, and functional architectures of the B4 software project.

Job Position Type	Responsibility
18. Design lead	Lead the software development process, implementation of code, testing strategies, and software deployment methods.
19. Requirements lead	Develop, review, analyze, and maintain software requirements for the B4 Project.
20. Programmer	Develop code and perform software integration activities.
21. Software installer	Develop software installation packages for deployment.
22. Test lead	Plan and manage all test related functions including test strategy, test plan development, test execution and reporting, development of personnel skills, and improvement of processes.
23. Configuration manager	Perform configuration management duties in support of software releases. Develop and maintain software version description documents, operational procedures, and internal documentation.
24. Product documentation lead	Develop software user's guide and technical manuals.
25. Standards compliance lead	Assess, review, and monitor process compliance in each program.
26. Tester	Test software and record software defects.
27. BFM lead	Monitor programs for project management requirements. Perform financial analysis and reporting.
28. Financial analyst	Provide guidance on policies and requirements of financial programs.
29. Financial technician	Prepare, process, and reconcile funding documents and related issues. Respond to data calls and cost estimating.
30. Contracting officer	Coordinate, implement, and monitor the contract compliance program. Provide technical advice and assistance in all areas of contracted support services.

Job Position Type	Responsibility
31. Contracting analyst	Provide advice and assistance to the contracting officer and program managers in cost and schedule contractual management, ensuring proper interpretations of reporting requirements.
32. Contract specialist	Negotiate sole source contracts and modifications. Provide guidance to technical personnel involved in the development of contract packages.
33. Contract representative	Maintain records of performance schedules and work progress reports. Monitor contractor performance and/or negotiate settlements.
34. Site manager	Lead, plan, execute, and conduct analyses concerning all aspects of services performed by contract.
35. Technical team lead	Serve as a supervisor of technical teams for contracting work.
36. Fleet liaison	Plan, coordinate, report, and execute fleet requirements. Analyze and resolve fleet-related issues.
37. Trainer	Conduct operational software training to fleet users. Test released software and report any software anomaly to product lead.
38. Fleet representative	Work closely with project managers and trainers in reviewing fleet requirements for the program.
39. Lead software engineer	Evaluate the effectiveness of DT/OT software function in addressing operational and information security requirements. Determine if a system is ready or not for fleet release.
40. Information assurance specialist	Evaluate, assess, analyze, and test software systems to ensure compliance with Information Assurance policies, instructions, and directives.

Job Position Type	Responsibility
41. Technical analyst	Assist lead software engineer to evaluate, assess, analyze, and test of DT/OT software.
42. Program analyst	Interface with customers from multiple technical teams and provide services required by customers.
43. Product integrity lead	Coordinate and distribute software to customers from multiple technical teams according to SDU release schedule and distribution list.
44. Systems engineer lead	Coordinate, assess, and monitor software release schedule for a specific program within the SDU.

7. B4 Project Complexity

Tables A-15, A-16, A-17, A-18, A-19, and A-20 show the scores and associated levels of complexity of six complexity metrics in the PCMM. We use a Likert scale [32] of 1 to 5 to map the scores of the PCMM metrics to the associated levels of complexity as shown in Tables A-15, A-16, A-17, A-18, A-19, and A-20.

Table A-15. The score and associated level of complexity of the measure of the number of personnel required for the B4 Project

Metric	PCMM value	Number of people per year can be funded (PMI)	Score	Complexity Level
1. Number of personnel required for the development effort (people)		1 to 7	1	Simple
		8 to 15	2	Complicated
	17.09	16 to 27	3	Low complexity
		28 to 34	4	Moderate complexity
		> 34 or unknown	5	High complexity

Table A-16. The score and associated level of complexity of the measure of defect density in the B4 Project

Metric	PCMM value	Defects / KLOC (software industry standard)	Score	Complexity Level
2. Defect density (defects / KLOC)	0.299	0 to 0.5	1	Simple
		0.51 to 1.1	2	Complicated
		1.2 to 2	3	Low complexity
		2.1 to 3	4	Moderate complexity
		> 3 or unknown	5	High complexity

Table A-17. The score and associated level of complexity of the measure of organizational complexity in the B4 Project

Metric	PCMM value	*IF* (Schwandt's study [158])	Score	Complexity Level
3. Organizational complexity		0 to 1.8	1	Simple
		1.9 to 3.7	2	Complicated
		3.8 to 5.5	3	Low complexity
	7.66	5.6 to 8.1	4	Moderate complexity
		More than 8.1 or unknown	5	High complexity

Table A-18. The score and associated level of complexity of the measure of geographical distribution of teams in the B4 Project

Metric	PCMM value	*GD* (guide from the Allen curve [119] and the IPT roles [197])	Score	Complexity Level
4. Geographical distribution of teams		1 to 3	1	Simple
		4 to 10	2	Complicated
	19	11 to 19	3	Low complexity
		20 to 40	4	Moderate complexity
		More than 40 or unknown	5	High complexity

Table A-19. The score and associated level of complexity of the measure of requirements volatility for the B4 Project

Metric	PCMM value	*RVM* (best practice in the software industry [162])	Score	Complexity Level
5. Requirements volatility		0 to 0.2	1	Simple
	0.388	0.21 to 0.42	2	Complicated
		0.43 to 0.66	3	Low complexity
		0.67 to 1	4	Moderate complexity
		More than 1 or unknown	5	High complexity

Table A-20. The score and associated level of complexity of the measure of the number of different job position types in the B4 Project

Metric	PCMM value	*JPT* (guide from the IPT roles [197])	Score	Complexity Level
6. Number of different job position types		1 to 15	1	Simple
		16 to 30	2	Complicated
	44	31 to 45	3	Low complexity
		46 to 65	4	Moderate complexity
		More than 65 or unknown	5	High complexity

In regard to project complexity, we have six metrics. The overall complexity level of the B4 Project is computed as the average of the six complexity scores of the six measures in the PCMM, which is 2.67 as shown in Table A-21. Hence, the complexity level of the B4 Project is low.

Table A-21. The complexity level of the B4 Project

Metric	PCMM value	Complexity Score	Complexity Level
1. Number of personnel required for the development effort (people)	17.05	3	Low
2. Defect density (defects / KLOC)	0.299	1	Simple
3. Organizational complexity	7.66	4	Moderate
4. Geographical distribution of teams	19	3	Low
5. Requirements volatility	0.388	2	Complicated
6. Number of different job position types	44	3	Low
Overall complexity level of the B4 Project		2.67	Low

In regard to project risk, we use a Likert scale [32] of 1 to 3 to map the total score from the six metrics of the PCMM to the associated risk level. Table A-22 shows the mapping between project complexity and risk level.

Table A-22. Mapping between project complexity and risk level

Project Complexity	Risk Level
1 – Simple	1 - Low
2 – Complicated	
3 – Low complexity	
4 – Moderate complexity	2 - Medium
5 – High complexity	3 - High

We rate the risk level of the project based on degree of impact in terms of cost, schedule, and performance on the project, as shown in Table A-23.

Table A-23. Rating of project risk level

Risk Level	Project Cost	Project Schedule	Project Performance
Low	A cost increase of less than 10%	A schedule slip of less than 2 weeks	Requirements can be met.
Medium	A cost increase between 10% to 20%	A schedule slip of between 2 to 4 weeks	Requirements can be met.
High	A cost increase of greater than 20%	A schedule slip of greater than 4 weeks	Requirements cannot be met. Major system redesign is required.

We use Table A-22 to score the six metrics in Tables A-15, A-16, A-17, A-18, A-19, and A-20. The overall risk level of the B4 Project is calculated as the average of the six risk scores of the six measures in the PCMM as shown in Table A-24, which is 1.167. Thus, the overall risk level of the B4 Project is low.

Table A-24. Risk level of each metric in the B4 Project

Metric	Risk Score	Risk Level
1. Number of personnel required for the development effort (people)	1	Low
2. Defect density (defects / KLOC)	1	Low
3. Organizational complexity	2	Medium
4. Geographical distribution of teams	1	Low
5. Requirements volatility	1	Low
6. Number of different job position types	1	Low
Overall risk level of the B4 Project	1.167	Low

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. COMPLEXITY MEASURES OF THE B6 PROJECT

In this appendix, we show the complexity measurements of the B6 Project, which include six measurements: (1) number of personnel required for the development effort, (2) defect density, (3) organizational complexity, (4) geographical distribution of teams, (5) requirements volatility, and (6) number of different job position types. Based on the PMA's project definitions (project cost and duration, capability development, and performance), the program manager's opinions regarding the statement of requirement, the number of personnel required in program development, project deliverables and schedules, and the empirical data of both projects B4 and B6, the B6 Project is deemed more complex than the B4 Project. The project plan shows the nine organizational units involved in milestone reviews and status reviews during the 28 months development phase, as shown in Table B-1.

Table B-1. Planned reviews listed in the B6 Project plan

Organizational Unit	Review	Frequency
PMA	CDP review	One time during the planning phase
IPT	Senior management review	Quarterly
	Project management review	Once every two weeks during the planning and development phase
PT	• System and software requirement review • Preliminary design review • Critical design review	Two times during the development phase
	Technical interchange meeting	Monthly
	Software engineering group review	Once every two weeks

Organizational Unit	Review	Frequency
	Project milestone review	Three times during the development phase
	CCB meeting	Once every two weeks

1. Measure of the Number of Personnel Required for the Development Effort

Table B-2 shows the calculation of software size for the B6 Project.

Table B-2. Calculation of software size for the B6 Project

Software size's driver [24]	B6 Project software size
Requirements	(282 difficult requirements)(5) = 1,410 (282 nominal requirements)(1) = 282 (2,258 easy requirements)(0.5) = 1,129 Total is 2,821 (1,410 + 282 + 1,129)
Interfaces	(7 easy interfaces)(1.7) = 11.9 (6 nominal interfaces)(4.3) = 25.8 (4 difficult interfaces)(9.8) = 39.2 Total is 76.9 (10.2 + 12.9 + 19.6)
Algorithms	(3 easy algorithms)(3.4) = 10.2 (2 nominal algorithms)(6.5) = 13 (2 difficult algorithms)(18.2) = 36.4 Total is 59.6 (10.2 + 13 + 36.4)
Operational Scenarios (use cases)	(29 easy use cases)(9.8) = 284.2 (6 nominal use cases)(22.8) = 136.8 (4 difficult use cases)(47.4) = 189.6 Total is 610.6 (284.2 + 136.8 + 189.6)
Total software size [24] = requirements + interfaces + algorithms + user cases	2,821 + 76.9 + 59.6 + 610.6 = 3,568.1 person-month

Table B-3 shows the calculation of the B6 Project's effort weight factor.

Table B-3. The B6 Project's effort weight factor

Effort weight factors	Value
Requirements understanding is high [24].	0.77
Technical risk is nominal [24].	1
Process capability is nominal [24].	1
The B6 Project's effort weight factor is calculated.	(0.77)(1)(1) = 0.77

Table B-4 shows the calculation of *PM* (person-month) [24] and the number of people required for the development effort. Note that the calibration constant is 0.325, and the effort weight factor is 0.77 in a typical software development project [24]. The B6 Project requires 31.89 people to develop the system in 28 months.

Table B-4. The B6 Project development effort

B6 Project development effort	Value
Effort required to build the system = *PM* = (calibration factors)(effort factor)(software size)	(0.325)(0.77)(3,568.1) = 892.92 person-month
B6 Project development effort = (effort required to build the system) / (project duration)	892.92 / 28 = 31.89 people

2. Defect Density Measure

Table B-5 shows the B6 Project defect density during the entire 28 months of the development phase. The defect density of the B6 Project is 1.78 in the first 28 months

(February 2017 to June 2019). The total test hours for that period of 28 months is 6,384.85 hours. 1,408 defects are reported in the 28-month period.

Table B-5. B6 Project's defect density during the first 28 months of the development phase.

Release (*i*)	**Month**	**Cumulative defects open (ND_i)**	**KLOC**	**Defect density (ND_i/KLOC)**	**Test Hours**
0	Feb-17	53	495.289	0.107	41
1	Mar-17	60	526.648	0.113	53
2	Apr-17	86	151.192	0.568	69
3	May-17	123	355.636	0.345	55.5
4	Jun-17	174	482.089	0.360	124
5	Jul-17	230	251.791	0.913	114
6	Aug-17	259	244.945	1.057	73
7	Sep-17	283	268.905	3.339	84.75
8	Oct-17	299	312.894	0.955	99
9	Nov-17	348	336.775	1.033	118.75
10	Dec-17	429	329.711	1.301	103.5
11	Jan-18	498	365.653	1.361	132.25
12	Feb-18	515	393.651	1.308	14.75
13	Mar-18	592	457.078	1.295	301.75
14	Apr-18	633	524.784	1.206	142.5
15	May-18	738	526.934	1.400	335
16	Jun-18	794	532.500	1.491	276
17	Jul-18	856	544.476	1.572	370.2
18	Aug-18	901	556.425	1.619	282.5
19	Sep-18	982	532.407	1.844	286.6
20	Oct-18	1062	492.262	2.157	474
21	Nov-18	1100	481.926	2.282	235.5
22	Dec-18	1137	475.657	2.390	212.8
23	Jan-19	1222	676.197	1.807	523.95
24	Feb-19	1273	675.668	1.884	446.55
25	Mar-19	1304	676.831	1.926	275
26	Apr-19	1340	677.286	1.978	460.5
27	May-19	1388	789.210	1.758	382
28	Jun-19	1408	790.487	1.781	297.5
Total					6,384.85

3. Organizational Complexity Measure

Figure B-1 shows the context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PMA and IPT, between PMA

and PT, and between PMA and the DT/OT unit during the 28 months development phase. For example, as shown in Figure B-1, IPT submits CDPs to the PMA one time with the communication level of importance of 5. PMA provides the program priorities to the IPT 28 times with the communication level of importance of 3 each time.

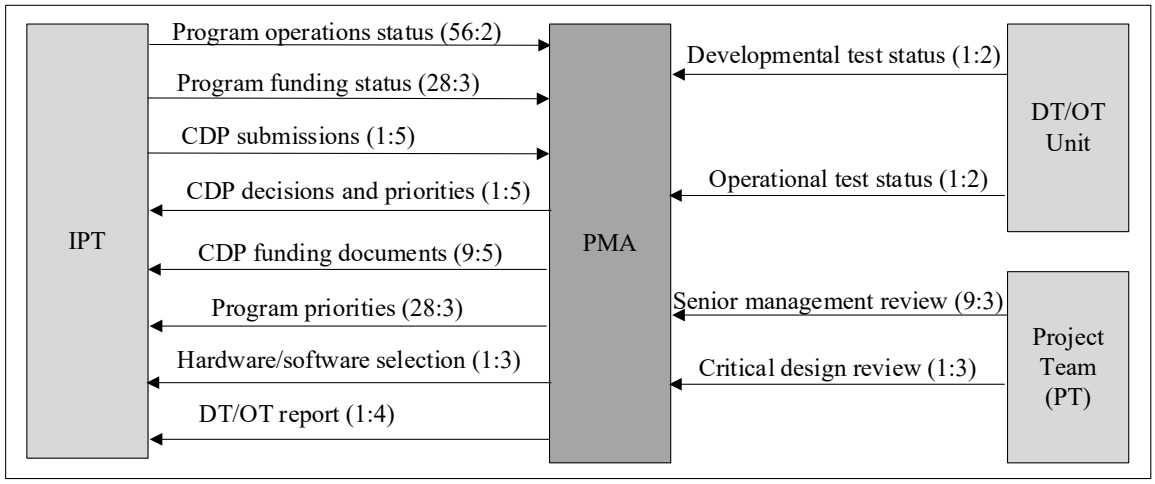

Figure B-1. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PMA and IPT, between PMA and PT, and also between PMA and the DT/OT unit

Figures B-2, B-3, and B-4 show the context diagrams of the rest of the nodes, links, the communication frequencies, and levels of importance of the communications between the organizations of the B6 Project.

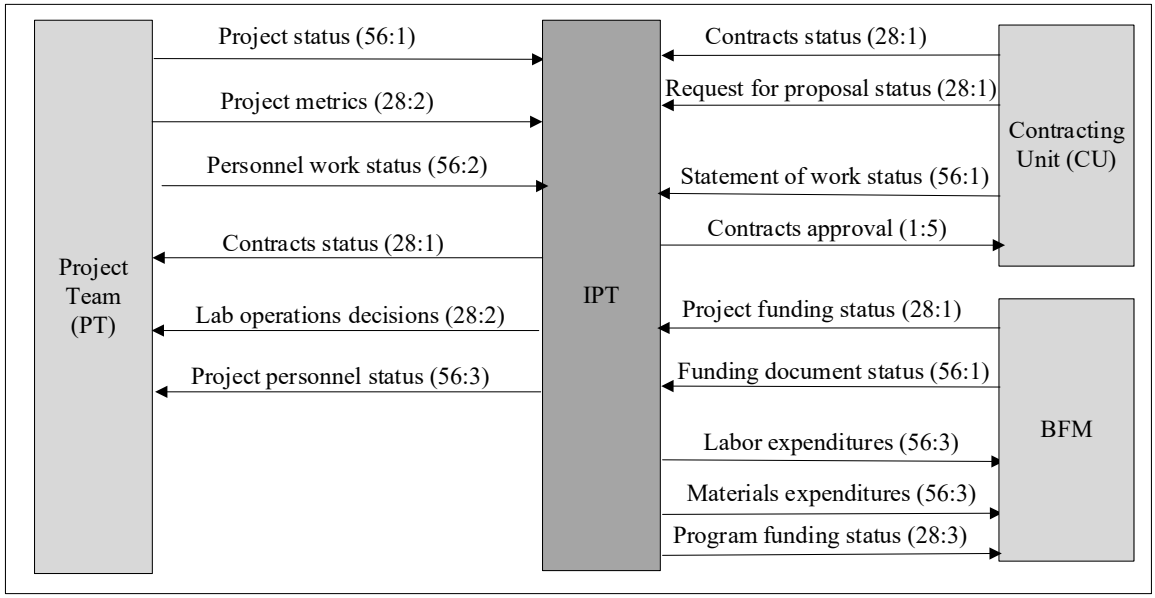

Figure B-2. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between IPT and PT, between IPT and CU, and between IPT and BFM

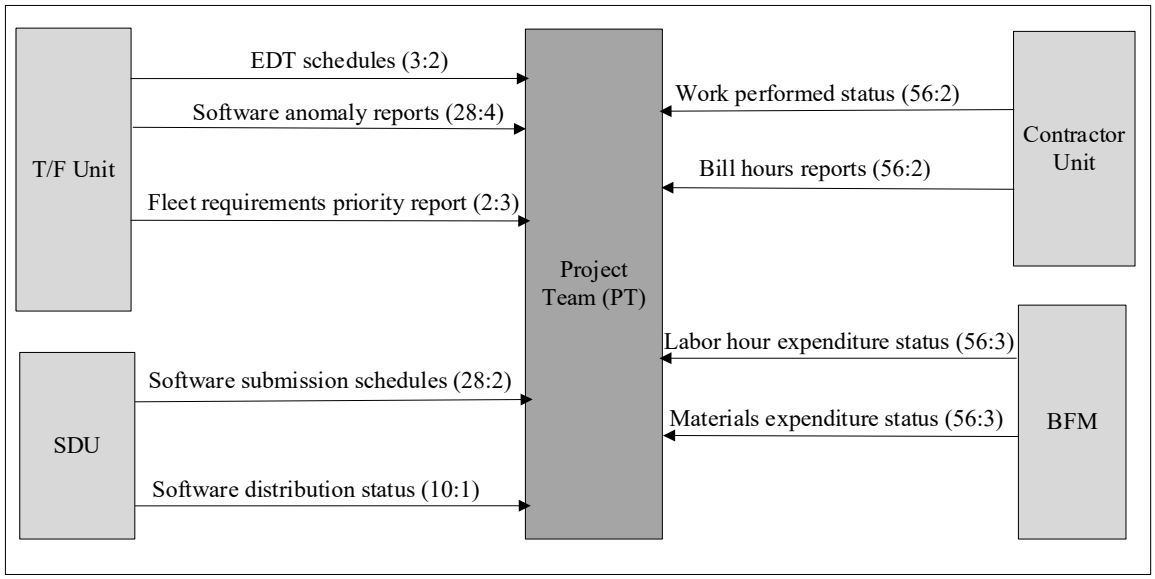

Figure B-3. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between PT and the T/F unit, between PT and SDU, between PT and BFM, and between PT and the contractor unit

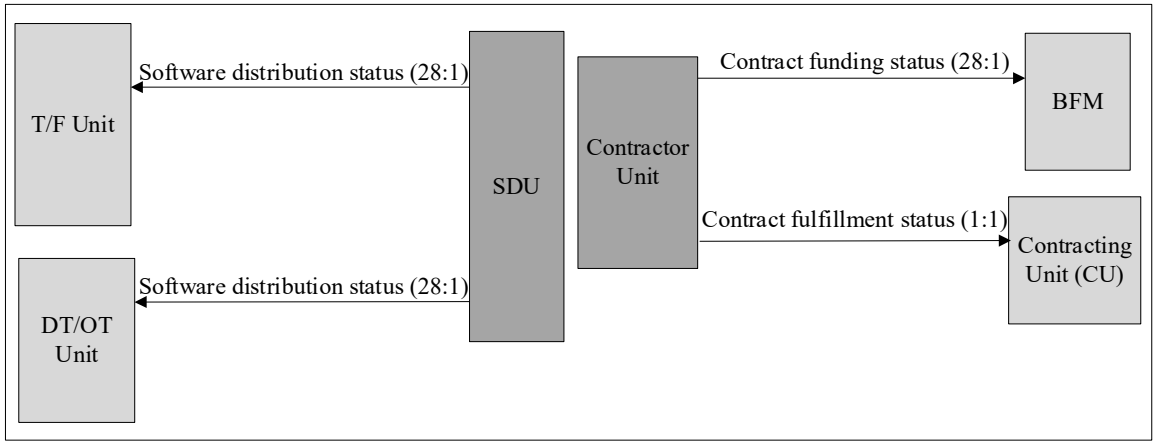

Figure B-4. A context diagram of nodes, links, communication frequencies, and levels of importance of the communications between SDU and the T/F unit, between SDU and DT/OT unit, between the contractor unit and BFM, and between the contractor unit and CU

Tables B-5 and B-6 show the frequencies and levels of importance of the communications in the B6 Project during the 28 months of the development phase. These data are extracted from project status reports, test reports, senior management reviews, and the project plan.

Table B-5. Frequencies of the communications during the first 28 months of the development phase

	Communication of information	**Frequency**						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
1. IPT	1. Program operations status		56					
	2. Program funding status		28					
	3. CDP submission		1					
	1. Contracts status			28				
	2. Lab operations status			28				

	Communication of information	Frequency						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
	3. Project personnel status			56				
	1. Labor expenditures				56			
	2. Materials expenditures				56			
	3. Program funding status				28			
	1. Contracts approval decisions					1		
	Subtotal		85	112	140	1		
2. PMA	1. CDP decisions	1						
	2. CDP funding documents	9						
	3. Program priorities status	28						
	4. B6 hardware/software selection	1						
	5. B6 DT/OT report	1						
	Subtotal	40						
3. Project Team (PT)	1. Project status	56						
	2. Project metrics	28						
	3. Personnel work status	56						
	1. Critical design review status		1					
	2. Senior management review		9					
	Subtotal	140	10					
4. BFM	1. Project funding status	28						
	2. Funding document status	56						
	1. Labor hour expenditure status			56				
	2. Materials expenditure status			56				
	Subtotal	84		112				
	1. Contracts status	28						

	Communication of information	Frequency						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
5. Contracting Unit (CU)	2. RFP status	28						
	3. Statement of work status	56						
	Subtotal	112						
6. Contractor Unit	1. Work performed status			56				
	2. Bill hours report			56				
	1. Contract funding status				28			
	1. Contract fulfillment status					1		
	Subtotal			112	28	1		
7. T/F Unit	1. EDT schedule			3				
	2. Software anomaly report			28				
	3. Fleet requirements priority report			2				
	Subtotal			33				
8. DT/OT Unit	1. DT status		1					
	2. OT status		1					
	Subtotal		2					
9. SDU	1. Software submission schedule			28				
	2. Software distribution status			10				
	1. Software release status						28	28
	Subtotal			38			28	28
	Total	376	97	407	168	2	28	28

Table B-6. Levels of importance of the communications during the first 28 months of the development phase

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
1. IPT	1. Program operations status		2					
	2. Program funding status		3					

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
	3. CDP submission		5					
	1. Contracts status			1				
	2. Lab operations status			2				
	3. Project personnel status			3				
	1. Labor expenditures				3			
	2. Materials expenditures				3			
	3. Program funding status				3			
	1. Contracts approval decisions					5		
2. PMA	1. CDP decisions	5						
	2. CDP funding documents	5						
	3. Program priorities status	3						
	4. Hardware/software decisions	3						
	5. DT/OT report	4						
3. Project Team	1. Project status	1						
	2. Project metrics	2						
	3. Personnel work status	2						
	1. Critical design review status		3					
	2. Senior management review		3					
4. BFM	1. Project funding status	1						
	2. Funding document status	1						
	1. Labor hour expenditure status			3				
	2. Materials expenditure status			3				
5. Contracting Unit (CU)	1. Contracts status	1						
	2. RFP status	1						
	3. Statement of work status	1						

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/OT Unit
6. Contractor Unit	1. Work performed status			2				
	2. Bill hours report			2				
	1. Contract funding status				1			
	1. Contract fulfillment status					1		
7. T/F Unit	1. EDT schedule			2				
	2. Software anomaly reporting			4				
	3. Fleet requirements priority report			3				
8. DT/OT Unit	1. DT status		2					
	2. OT status		2					
9. SDU	1. Software submission schedule			2				
	2. Software distribution status			1			1	1

Table B-7 shows the calculations of organizational complexity of each organizational unit. We sum up the contribution of organizational complexity from each of the nine organizational units during the period of 28 months and obtain the total organizational complexity of 7.64.

Table B-7. Organizational complexity of each organizational unit in the B6 Project

Organizational Unit	*IF*				Total
1. IPT interacts with PMA, PT, BFM, and CU.	PMA: [(2)(56)+(3)(28)+(5)(1)]/[(5)(85)]= 0.473	PT: [(1)(28)+(2)(28)+(3)(56)]/[(5)(112)]=0.45	BFM: [(3)(56)+(3)(56)+(3)(28)]/[(5)(140)]=0.6	CU: [(1)(5)]/[(5)(1)] =1	2.52
2. PMA interacts with IPT.	IPT: [(5)(1)+(5)(9)+(3)(28)+(3)(1)+(4)(1)]/[(5)(40)]=0.705				0.705

Organizational Unit	*IF*			Total
3. PT interacts with IPT and PMA.	IPT: [(1)(56)+(2)(28)+(2)(56)]/[(5)(140)]= 0.32		PMA: [(3)(1)+(3)(9)]/[(5)(10)]=0.6	0.92
4. BFM interacts with IPT and PT.	IPT: [(1)(28)+(1)(56)]/[(5)(81)]=0.2		PT: [(2)(56)+(2)(56)]/[(5)(112)]=0.4	0.6
5. CU interacts with IPT.	IPT: [(1)(28)+(1)(28)+(1)(56)]/[(5)(112)]=0.2			0.2
6. Contractor Unit interacts with PT, BFM, and CU.	PT: [(2)(56)+(2)(56)]/[(5)(112)]=0.4	BFM: [(1)(28)]/[(5)(28)]=0.2	CU: [(1)(1)]/[(5)(1)]=0.2	0.8
7. T/F Unit interacts with PT.	PT: [(2)(3)+(4)(28)+(3)(2)]/[(5)(33)]=0.751			0.751
8. DT/OT Unit interacts with PMA.	PMA: [(2)(1)+(2)(1)]/[(5)(2)]=0.4			0.4
9. SDU interacts with PT, T/F Unit, and DT/OT Unit.	PT: [(2)(28)+(1)(10)]/[(5)(38)]=0.347	T/F Unit: [(1)(28)]/[(5)(28)]=0.2	DT/OT Unit: [(1)(28)]/[(5)(28)]=0.2	0.747
Total Contribution				7.64

4. Geographical Distribution of Teams Measure

Table B-8 shows the geographical distribution of teams from the nine organizational units in the B6 Project. We sum up the number of sites, locations, and time zones. We obtain the value of 20 for the measure of geographical distribution of teams.

Table B-8. Geographical distributions of teams in the B6 Project

	Number of Sites	**Location**	**Time Zone**
1. PMA	1	East-coast city	Eastern time
2. IPT	1	Southwest-1city	Pacific time

	Number of Sites	Location	Time Zone
3. PT	4	Southwest-1 city	Pacific time
4. BFM	1	Southwest-1 city	Pacific time
5. CU	1	Southwest-1 city	Pacific time
6. Contractor Unit	2	Southwest-1 city	Pacific time
7. T/F Unit	1	Northwest city	Pacific time
8. DT/OT Unit	2	East-coast city and Southwest-2 city	Eastern time and Pacific time
9. SDU	1	Southwest-1 city	Pacific time
Total	14	4	2

5. Requirements Volatility Measure

Table B-9 show the requirements metrics during the 28 months of the development phase. As presented in Table B-9, the project starts with a baseline of 2,089 requirements. During a 28-month period, the B6 Project has 2,725 change requests (1,586 new + 286 modifications + 853 deletions) and 2,822 requirements.

Table B-9. B6 Project requirements metrics during first 28 months of the development phase

	Requirement (CR_i)			BR = 2,089 requirements	
Baseline (i)	New	Modification	Deletion	Total # of requirements	Requirements volatility (CR_i)/(BR)
0				2,089	0
1	10		30	2,069	0.019
2	6		14	2,061	0.009
3	19		8	2,072	0.013
4	260	4	45	2,287	0.148
5	5			2,292	0.002
6	46	1	9	2,329	0.027
7	169	4	37	2,461	0.1
8	26		4	2,483	0.014
9	3	53	3	2,483	0.028
10	21		18	2,486	0.018
11	226	42	15	2,697	0.135
12	268	29	100	2,865	0.19
13	34	27	64	2,835	0.059
14	23		53	2,805	0.036
15	107		138	2,774	0.117
16	23	5	40	2,757	0.032
17	31		4	2,784	0.016
18	24		59	2,749	0.039
19	37	4	27	2,759	0.032
20	17	14	8	2,768	0.018
21	34	8	85	2,717	0.061
22	37	7	21	2,733	0.031
23	1	1	1	2,733	0.001
24	20	1	17	2,736	0.018
25	137	86	39	2,834	0.125
26	2		14	2,822	0.007
Total	1,586	286	853		

6. Number of Different Job Position Types

The B6 Project plan shows 44 types of job positions that are identical to those in the B4 Project, as shown in Table A-14, Appendix A. Thus, the measure of different job position types (*JPT*) is 44.

7. B6 Project Complexity

Tables B-10, B-11, B-12, B-13, B-14, and B-15 show the scores and associated levels of complexity of six complexity metrics in the PCMM. Similar to the B4 Project, we

use a Likert scale [32] of 1 to 5 to map the scores of the PCMM metrics to the associated levels of complexity as shown in Tables B-10, B-11, B-12, B-13, B-14, and B-15.

Table B-10. The score and associated level of complexity of the measure of the number of personnel required for the B6 Project

Metric	PCMM value	Number of people per year can be funded (PMI)	Score	Complexity Level
1. Number of personnel required for the development effort (people)		1 to 7	1	Simple
		8 to 15	2	Complicated
		16 to 27	3	Low complexity
		28 to 34	4	Moderate complexity
	35.08	> 34 or unknown	5	High complexity

Table B-11. The score and associated level of complexity of the measure of defect density in the B6 Project

Metric	PCMM value	Defects / KLOC (software industry standard)	Score	Complexity Level
2. Defect density (defects / KLOC)		0 to 0.5	1	Simple
		0.51 to 1.1	2	Complicated
	1.78	1.2 to 2	3	Low complexity
		2.1 to 3	4	Moderate complexity
		> 3 or unknown	5	High complexity

Table B-12. The score and associated level of complexity of the measure of organizational complexity in the B6 Project

Metric	PCMM value	*IF* (Schwandt's study [158])	Score	Complexity Level
3. Organizational complexity		0 to 1.8	1	Simple
		1.9 to 3.7	2	Complicated
		3.8 to 5.5	3	Low complexity
	7.64	5.6 to 8.1	4	Moderate complexity
		More than 8.1 or unknown	5	High complexity

Table B-13. The score and associated level of complexity of the measure of geographical distribution of teams in the B6 Project

Metric	PCMM value	*GD* (guide from the Allen curve [119] and the IPT roles [197])	Score	Complexity Level
4. Geographical distribution of teams		1 to 3	1	Simple
		4 to 10	2	Complicated
		11 to 19	3	Low complexity
	20	20 to 40	4	Moderate complexity
		More than 40 or unknown	5	High complexity

Table B-14. The score and associated level of complexity of the measure of requirements volatility in the B6 Project

Metric	PCMM value	*RVM* (best practice in the software industry [162])	Score	Complexity Level
5. Requirements volatility		0 to 0.2	1	Simple
		0.21 to 0.42	2	Complicated
		0.43 to 0.66	3	Low complexity
		0.67 to 1	4	Moderate complexity
	1.3	More than 1 or unknown	5	High complexity

Table B-15. The score and associated level of complexity of the measure of the number of different job position types in the B6 Project

Metric	PCMM value	*JPT* (guide from the IPT roles [197])	Score	Complexity Level
6. Number of different job position types		1 to 15	1	Simple
		16 to 30	2	Complicated
	44	31 to 45	3	Low complexity
		46 to 65	4	Moderate complexity
		More than 65 or unknown	5	High complexity

The overall complexity level of the B6 Project is computed as the average complexity scores of the six measures in the PCMM as shown in Tables B-16. With the average complexity scores of 4, the overall complexity level of the B6 Project is moderate, as shown in Table B-16.

Table B-16. The complexity level of the B6 Project

Metric	PCMM value	Complexity Score	Complexity Level
1. Number of personnel required for the development effort (people)	35.08	5	High
2. Defect density (defects / KLOC)	1.78	3	Low
3. Organizational complexity	7.64	4	Moderate
4. Geographical distribution of teams	20	4	Moderate
5. Requirements volatility	1.3	5	High
6. Number of different job position types	44	3	Low
Overall complexity level of the B6 Project		4	Moderate

We use Table A-22 to score the six metrics in Tables B-10, B-11, B-12, B-13, B-14, and B-15. The overall risk level of the B6 Project is calculated as the average of the six risk scores of the six measures in the PCMM, which is 2 as shown in Table B-17. Thus, the overall risk level of the B6 Project is medium as shown in Table B-17.

Table B-17. The risk level of each metric in the B6 Project

Metric	Score	Risk Level
1. Number of personnel required for the development effort (people)	3	High
2. Defect density (defects / KLOC)	1	Low
3. Organizational complexity	2	Medium
4. Geographical distribution of teams	2	Medium
5. Requirements volatility	3	High
6. Number of different job position types	1	Low
Overall risk level of the B6 Project	2	Medium

APPENDIX C. COMPLEXITY MEASURES OF THE B8 PROJECT

In this appendix, we present the complexity measurements of the B8 Project. These measurements include the number of personnel required for the development effort, defect density, organizational complexity, geographical distribution of project teams, requirements volatility, and the number of different job position types. Based on the PMA's project definitions, the program manager's opinion, and the empirical data from the project profiles B4, B6, and B8, the B8 Project is less complex than the B6 Project but more complex than the B4 Project. The B8 Project plan listed stakeholders' roles and functions, as shown in Table C-1.

Table C-1. Stakeholders' roles and functions listed in the B8 Project plan

Stakeholder Role	Function
Program sponsor	• Define future operational requirements for the fleet. • Set program's priorities and allocate funding.
Program manager	• Develop the project schedule, budget, and evaluate the technical performance of the SoI. • Approve or disapprove milestone reviews. • Oversee procurements.
Product lead	• Oversee system definition, development, integration, and testing. • Establish and maintain the project management plan (PMP). • Create program increment plan (PIP) and conduct program increment planning meetings. • Interface with IPT and external organizations for project status. • Participate in milestone reviews and IPT management reviews.

Stakeholder Role	Function
Senior management	• Provide personnel and lab resources to each project. • Provide resources for personnel development and training. • Commit to IPT policies, process improvement, and industrial engineering development standards.
BFM office	• Allocate funding within the IPT. • Provide contractor oversight. • Oversee IPT procurements. • Advise senior management on any anomalies.
Fleet representation	• Provide operational insight and end user direction during the development life cycle for new capabilities and modifications.
Naval aviation depot	• Generate hardware change packages for the specified aircraft.
Developmental testing	• Verify and validate that all new system requirements are met. • Perform a regression test to verify and validate that previously existing requirements still perform correctly by the system.
Operational testing	• Test new or modified systems as they would be used in the fleet. • Determine whether a system is ready for fleet release.

The project plan also shows the nine organizational units involved in milestone reviews and status reviews during the 20 months development phase, as shown in Table C-2.

Table C-2. Planned reviews from the B8 Project plan

Organizational Unit	Review	Frequency
PMA	• CDP review	One time during the planning phase
IPT	• Senior management review	Quarterly
	• Project management review	Once every two weeks during the planning and development phase
PT	• System and software requirement review • Preliminary design review • Critical design review	Two times during the development phase
	• Technical interchange meeting	Monthly
	• Software engineering group review	Once every two weeks
	• Project milestone review	Three times during the development phase
	• CCB meeting	Once every two weeks

1. Measure of the Number of Personnel Required for the Development Effort

Table C-3 shows the calculation of software size for the B8 Project.

Table C-3. Calculation of software size for the B8 Project

Software size's driver [24]	B8 Project software size
Requirements	(578 nominal requirements)(1) = 578 (2,313 easy requirements)(0.5) = 1,156.5 Total is 1,734.5 (578 + 1,156.5).
Interfaces	(2 easy interfaces)(1.7) = 3.4 Total is 3.4.
Algorithms	(1 easy algorithms)(3.4) = 3.4 Total is 3.4.
Operational Scenario (use cases)	(38 easy use cases)(9.8) = 372.4 (5 nominal use cases)(22.8) = 114 Total is 486.4 (372.4 + 114).
Total software size [24] = requirements + interfaces + algorithms + user cases	1,734.5 + 3.4 + 3.4 + 486.4 = 2,227.7

Table C-4 shows the computation of the B8 Project's effort weight factor.

Table C-4. The B8 Project's effort weight factor

Effort weight factors	**Value**
Requirements understanding is high [24].	0.77
Technical risk is nominal [24].	1
Process capability is nominal [24].	1
The B6 Project's effort weight factor is calculated	(0.77)(1)(1) = 0.77

Table C-5 shows the calculation of *PM* (person-month) [24] and the number of people required for the development effort. Note that the calibration constant is 0.325, and the effort weight factor is 0.77 in a typical software development project [24]. The B8 Project requires 27.87 people to develop the system in 20 months.

Table C-5. The B8 Project development effort

B8 Project development effort	Value
Effort required to build the system = *PM* = (calibration factors)(effort factor)(software size)	(0.325)(0.77)(2,227.7) = 557.48 person-month
B6 Project development effort = (effort required to build the system) / (project duration)	557.48 / 20 = 27.87 people

2. Defect Density Measure

Table C-6 shows the B8 Project's defect density during the first 20 months of the development phase. After the first 20 months in the development phase (Jun-21), the project averaged 0.109 defects per KLOC, which is well within the industry standard limit of 3. The total test hours for the 20-month period is 838.5 hours. There are 49 defects reported in the 20-month period.

In Table C-6, the defect density trend is upward toward the end of the first 20 months development cycle (October 2019 through June 2021), indicating that test hours have increased dramatically before the release of the software to stakeholders. This means that either the testers spent fewer hours testing and closing the open defects, or they spent more hours testing the software and found more defects.

Table C-6. B8 Project's defect density during the first 20 months of the development phase

Release	Month	Cumulative defects open (ND_i)	KLOC	Defect density (ND_i/KLOC)	Test Hours
0	Oct-19	1	787.303	0.001	50
1	Nov-19	1	784.546	0.001	9
2	Dec-19	1	786.059	0.001	17
3	Jan-20	1	786.059	0.001	0
4	Feb-20	1	786.059	0.001	0
5	Mar-20	1	355.542	0.003	16
6	Apr-20	6	323.950	0.019	35
7	May-20	10	442.604	0.022	42
8	Jun-20	10	442.604	0.022	10
9	Jul-20	16	446.237	0.035	48
10	Aug-20	16	440.028	0.036	0
11	Sep-20	21	443.354	0.047	66
12	Oct-20	35	449.464	0.078	175
13	Nov-20	40	449.789	0.089	104
14	Dec-20	41	448.745	0.091	47
15	Jan-21	44	448.741	0.098	24
16	Feb-21	44	448.818	0.098	27.5
17	Mar-21	44	449.836	0.098	0
18	Apr-21	44	449.836	0.098	7
19	May-21	46	449.836	0.102	64
20	Jun-21	49	449.836	0.109	97
Total					838.5

3. Organizational Complexity Measure

Figure C-1 shows the context diagram of nodes, links, communication frequencies and levels of importance of communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit during the first 20-month development phase. For example, as shown in Figure C-1, IPT submits CDPs to the PMA one time with the communication level of importance of 5. PMA provides the program priorities to the IPT 20 times with the communication level of importance of 3 each time.

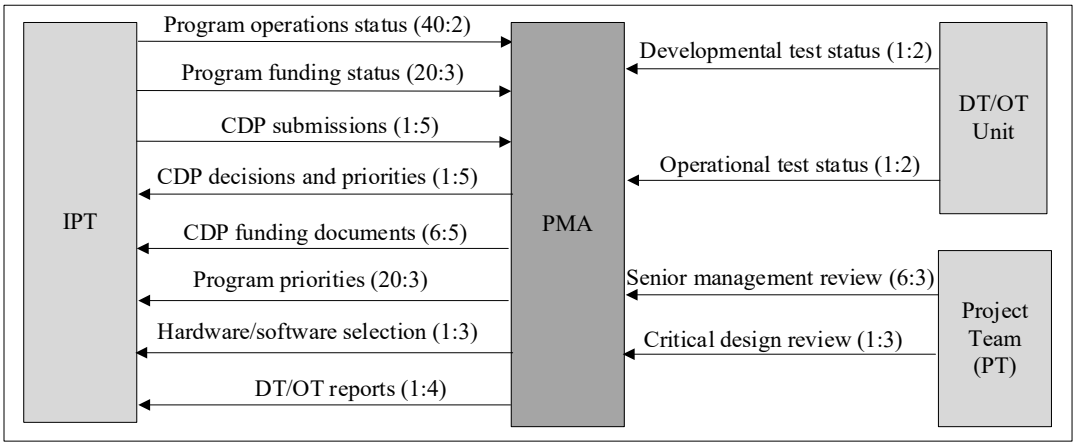

Figure C-1. A context diagram of nodes, links, communication frequencies, and importance levels of the communications between PMA and IPT, between PMA and PT, and between PMA and the DT/OT unit

Figures C-2, C-3, and C-4 show the context diagrams of the rest of the nodes, links, communication frequencies, and importance levels of the communications between the organizations of the B6 Project.

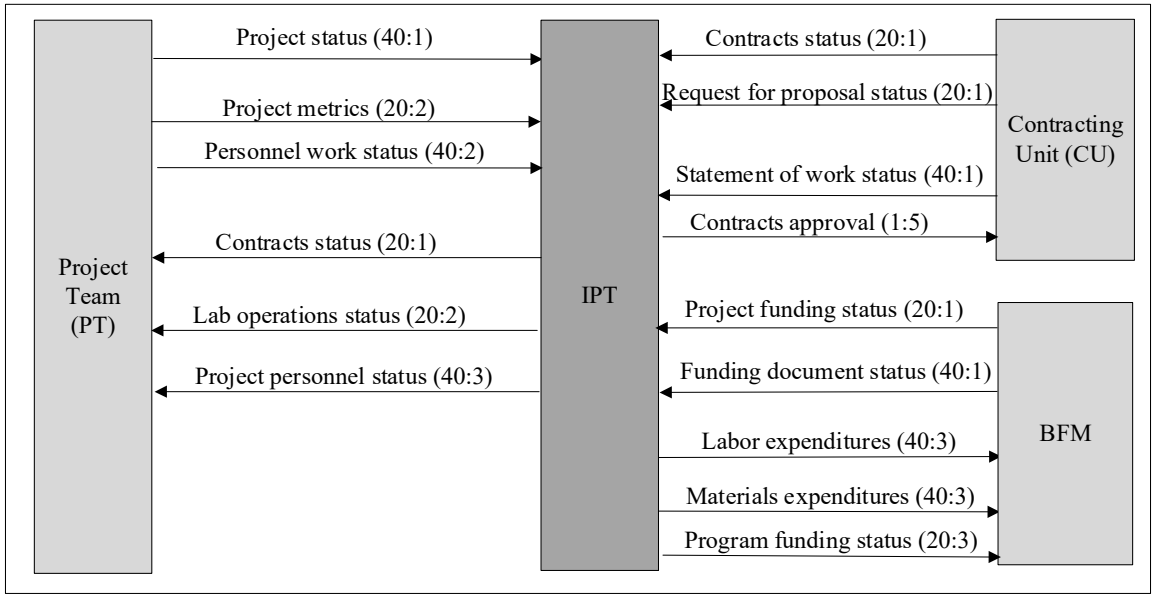

Figure C-2. A context diagram of nodes, links, communication frequencies, and importance levels of the communications between IPT and PT, between IPT and CU, and between IPT and BFM

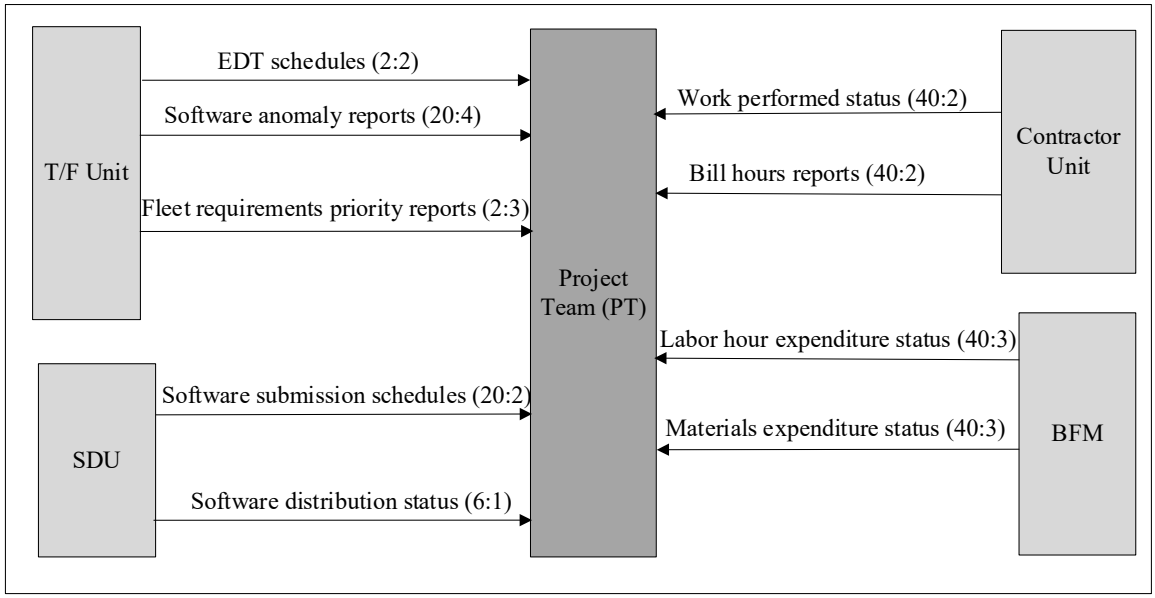

Figure C-3. A context diagram of nodes, links, communication frequencies, and importance levels of the communications between PT and the T/F unit, between PT and SDU, between PT and BFM, and between PT and the contractor unit

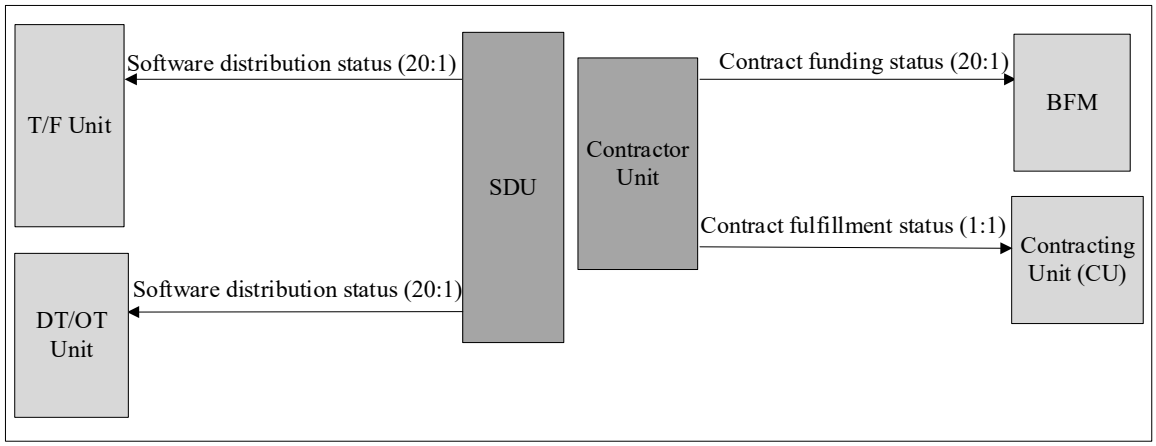

Figure C-4. A context diagram of nodes, links, communication frequencies, and importance levels of the communications between SDU and the T/F unit, between SDU and DT/OT unit, between the contractor unit and BFM, and between the contractor unit and CU

Tables C-7 and C-8 show the frequencies and levels of importance of the communications in the B8 Project during the 20 months of the development phase. These

data are extracted from project status reports, test reports, senior management reviews, and the project plan.

Table C-7. Frequencies of the communications during the first 20 months of the development phase

	Communication of information	**Frequency**						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
1. IPT	1. Program operations status		40					
	2. Program funding status		20					
	3. CDP submission		1					
	1. Contracts status			20				
	2. Lab operations status			20				
	3. Project personnel status			40				
	1. Labor expenditures				40			
	2. Materials expenditures				40			
	3. Program funding status				20			
	1. Contracts approval decisions					1		
	Subtotal		61	80	100	1		
2. PMA	1. CDP decisions	1						
	2. CDP funding documents	6						
	3. Program priorities status	20						
	4. hardware/software selection	1						
	5. DT/OT reports	1						
	Subtotal	29						
3. Project Team (PT)	1. Project status	40						
	2. Project metrics	20						
	3. Personnel work status	40						
	1. Critical design review status		1					

	Communication of information	Frequency						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
	2. Senior management reviews		5					
	Subtotal	100	6					
4. BFM	1. Project funding status	20						
	2. Funding document status	40						
	1. Labor hour expenditure status			40				
	2. Materials expenditure status			40				
	Subtotal	60		80				
5. Contracting Unit (CU)	1. Contracts status	20						
	2. RFP status	20						
	3. Statement of work status	40						
	Subtotal	80						
6. Contractor Unit	1. Work performed status			40				
	2. Bill hours reports			40				
	1. Contract funding status				20			
	1. Contract fulfillment status					1		
	Subtotal			80	20	1		
7. T/F Unit	1. EDT schedules			2				
	2. Software anomaly reports			20				
	3. Fleet requirements priority reports			2				
	Subtotal			24				
8. DT/OT Unit	1. DT status		1					
	2. OT status		1					
	Subtotal		2					
9. SDU	1. Software submission schedules			20				
	2. Software distribution status			6				
	1. Software release status						20	20

	Communication of information	Frequency						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
	Subtotal			26			20	20
	Total	269	69	290	120	2	20	20

Table C-8. Levels of importance of the communications during the first 20 months of the development phase

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
1. IPT	1. Program operations status		2					
	2. Program funding status		3					
	3. CDP submission		5					
	1. Contracts status			1				
	2. Lab operations status			2				
	3. Project personnel status			3				
	1. Labor expenditures				3			
	2. Materials expenditures				3			
	3. Program funding status				3			
	1. Contracts approval decisions					5		
2. PMA	1. CDP decisions	5						
	2. CDP funding documents	5						
	3. Program priorities status	3						
	4. Hardware/software decisions	3						
	5. DT/OT report	4						
3. Project Team	1. Project status	1						
	2. Project metrics	2						

	Communication of information	Level of importance						
		IPT	PMA	PT	BFM	CU	T/F Unit	DT/ OT Unit
	3. Personnel work status	2						
	1. Critical design review status		3					
	2. Senior management review		3					
4. BFM	1. Project funding status	1						
	2. Funding document status	1						
	1. Labor hour expenditure status			3				
	2. Materials expenditure status			3				
5. Contracting Unit (CU)	1. Contracts status	1						
	2. RFP status	1						
	3. Statement of work status	1						
6. Contractor Unit	1. Work performed status			2				
	2. Bill hours report			2				
	1. Contract funding status				1			
	1. Contract fulfillment status					1		
7. T/F Unit	1. EDT schedule			2				
	2. Software anomaly reporting			4				
	3. Fleet requirements priority report			3				
8. DT/OT Unit	1. DT status		2					
	2. OT status		2					
9. SDU	1. Software submission schedule			2				
	2. Software distribution status			1			1	1

Table C-9 shows the calculations of organizational complexity of each organizational unit. We sum up the contribution of organizational complexity from each of the nine organizational units during the period of 20 months and obtain the total organizational complexity of 7.65.

Table C-9. Organizational complexity of each organizational unit in the B8 Project

Organizational Unit	*IF*				Total
1. IPT interacts with PMA, PT, BFM, and CU.	PMA: [(2)(40)+(3)(20)+(5)(1)]/[(5)(61)]=0.475	PT: [(1)(20)+(2)(20)+(3)(40)]/[(5)(80)]=0.45	BFM: [(3)(40)+(3)(40)+(3)(20)]/[(5)(100)]=0.6	CU: (1)(5)/(5)(1) =1	2.52
2. PMA interacts with IPT.	IPT: [(5)(1)+(5)(6)+(3)(20)+(3)(1)+(4)(1)]/[(5)(29)]=0.703				0.703
3. PT interacts with IPT and PMA.	IPT: [(1)(40)+(2)(20)+(2)(40)]/[(5)(100)]=0.32		PMA: [(3)(1)+(3)(6)]/[(5)(7)]=0.6		0.92
4. BFM interacts with IPT and PT.	IPT: [(1)(20)+(1)(40)]/[(5)(60)]=0.2		PT: [(2)(20]+(2)(40)]/[(5)(60)]=0.4		0.6
5. CU interacts with IPT.	IPT: [(1)(20)+(1)(20)+(1)(40)]/[(5)(80)]=0.2				0.2
6. Contractor Unit interacts with PT, BFM, and CU.	PT: [(2)(40)+(2)(40)]/[(5)(80)]=0.4	BFM: [(1)(20)]/[(5)(20)]=0.2	CU: [(1)(1)]/[(5)(1)]=0.2		0.8
7. T/F Unit interacts with PT.	PT: [(2)(2)+(4)(20)+(3)(2)]/[(5)(24)]=0.75				0.75
8. DT/OT Unit interacts with PMA.	PMA: [(2)(1)+(2)(1)]/[(5)(2)]=0.4				0.4
9. SDU interacts with PT, T/F Unit, and DT/OT Unit.	PT: [(2)(20)+(1)(6)]/[(5)(26)]=0.354	T/F Unit: [(1)(20)]/[(5)(20)]=0.2	DT/OT Unit: [(1)(20)]/[(5)(20)]=0.2		0.754
Total Contribution					7.65

4. Geographical Distribution of Teams Measure

Table C-10 shows the geographical distribution of teams from the nine organizational units in the B8 Project. We sum up the number of sites, locations, and time zones. We obtain the value of 20 for the measure of geographical distribution of teams.

Table C-10. Geographical distributions of teams in the B8 Project

	Number of Sites	**Location**	**Time Zone**
1. PMA	1	East-coast city	Eastern time
2. IPT	1	Southwest-1city	Pacific time
3. PT	4	Southwest-1 city	Pacific time
4. BFM	1	Southwest-1 city	Pacific time
5. CU	1	Southwest-1 city	Pacific time
6. Contractor Unit	2	Southwest-1 city	Pacific time
7. T/F Unit	1	Northwest city	Pacific time
8. DT/OT Unit	2	East-coast city and Southwest-2 city	Eastern time and Pacific time
9. SDU	1	Southwest-1 city	Pacific time
Total	14	4	2

5. Requirements Volatility Measure

Table C-11 show the requirements metrics during the 20 months of the development phase. As presented in Table C-10, the project starts with a baseline of 2,822 requirements. During a 20-month period, the B8 Project has 1,029 change requests (533 new + 32 modifications + 464 deletions) and 2,822 requirements. The overall requirements volatility measure is computed as 0.36 (RVM = 1,029 / 2,822).

Table C-11. B8 Project requirement metrics during the first 20 months of the development phase

	Requirement (CR_i)			BR = 2,822 requirements	
Baseline	New	Modification	Deletion	Total # of requirements	Requirements volatility (CR_i/BR)
0				2,822	0
1	26			2,848	0.009
2	5			2,853	0.001
3	5		4	2,854	0.003
4	62		22	2,894	0.029
5	20		2	2,912	0.008
6	21		18	2,915	0.013
7	24	24	6	2,933	0.019
8	53	1	2	2,984	0.019
9	11		53	2,942	0.022
10	23		43	2,922	0.023
11	107		138	2,891	0.087
12	23	5	40	2,874	0.024
13	2		4	2,872	0.002
14	24		59	2,837	0.029
15	2		1	2,838	0.001
16	25		22	2,841	0.017
17	21		8	2,854	0.01
18	19	1	17	2,856	0.013
19	58	1	11	2,903	0.024
20	2		14	2,891	0.006
Total	533	32	464		

6. Number of Different Job Position Types

The B8 Project plan shows 44 types of job positions that are identical to those in the B4 Project, as shown in Table A-14, Appendix A. Thus, the measure of different job position types (*JPT*) is 44.

7. B8 Project Complexity

Tables C-12, C-13, C-14, C-15, C-16, and C-17 show the scores and associated levels of complexity of six complexity metrics in the PCMM. Similar to the B6 Project, we use a Likert scale [32] of 1 to 5 to map the scores of the PCMM metrics to the associated levels of complexity as shown in Tables C-12, C-13, C-14, C-15, C-16, and C-17.

Table C-12. The score and associated level of complexity of the measure of the number of personnel required for the B8 Project

Metric	PCMM value	Number of people per year can be funded (PMI)	Score	Complexity Level
1. Number of personnel required for the development effort (people)		1 to 7	1	Simple
		8 to 15	2	Complicated
		16 to 27	3	Low complexity
	30.66	28 to 34	4	Moderate complexity
		> 34 or unknown	5	High complexity

Table C-13. The score and associated level of complexity of the measure of defect density in the B8 Project

Metric	PCMM Value	Defects / KLOC (software industry standard)	Score	Complexity Level
2. Defect density (defects / KLOC)	0.109	0 to 0.5	1	Simple
		0.51 to 1.1	2	Complicated
		1.2 to 2	3	Low complexity
		2.1 to 3	4	Moderate complexity
		> 3 or unknown	5	High complexity

Table C-14. The score and associated level of complexity of the measure of organizational complexity in the B8 Project

Metric	PCMM value	*IF* (Schwandt's study [158])	Score	Complexity Level
3. Organizational complexity		0 to 1.8	1	Simple
		1.9 to 3.7	2	Complicated
		3.8 to 5.5	3	Low complexity
	7.65	5.6 to 8.1	4	Moderate complexity
		More than 8.1 or unknown	5	High complexity

Table C-15. The score and associated level of complexity of the measure of geographical distribution of teams in the B8 Project

Metric	PCMM value	*GD* (guide from the Allen curve [119] and the IPT roles [197])	Score	Complexity Level
4. Geographical distribution of teams		1 to 3	1	Simple
		4 to 10	2	Complicated
		11 to 19	3	Low complexity
	20	20 to 40	4	Moderate complexity
		More than 40 or unknown	5	High complexity

Table C-16. The score and associated level of complexity of the measure of requirements volatility in the B8 Project

Metric	PCMM Value	*RVM* (best practice in the software industry [162])	Score	Complexity Level
5. Requirements volatility		0 to 0.2	1	Simple
	0.36	0.21 to 0.42	2	Complicated
		0.43 to 0.66	3	Low complexity
		0.67 to 1	4	Moderate complexity
		More than 1 or unknown	5	High complexity

Table C-17. The score and associated level of complexity of the measure of number of different job position types in the B8 Project

Metric	PCMM value	*JPT* (guide from the IPT roles [197])	Score	Complexity Level
6. Number of different job position types		1 to 15	1	Simple
		16 to 30	2	Complicated
	44	31 to 45	3	Low complexity
		46 to 65	4	Moderate complexity
		More than 65 or unknown	5	High complexity

The overall project complexity of the B8 Project is computed as the average complexity scores of the six measures in the PCMM as shown in Tables C-18. With the average complexity scores of 3, the overall complexity level of the B8 Project is low as shown in Table C-18.

Table C-18. The complexity level of the B8 Project

Metric	PCMM value	Complexity Score	Complexity Level
1. Number of personnel required for the development effort (people)	30.66	4	Moderate
2. Defect density (defects / KLOC)	0.109	1	Simple
3. Organizational complexity	7.65	4	Moderate
4. Geographical distribution of teams	20	4	Moderate
5. Requirements volatility	0.36	2	Complicated
6. Number of different job position types	44	3	Low
Overall complexity level of the B8 Project		3	Low

We use Table A-22 to score the six metrics in Tables C-12, C-13, C-14, C-15, C-16, and C-17. The overall risk level of the B8 Project is computed as the average of the six risk scores of the six measures in the PCMM, which is 1.5 as shown in Table C-19. With the overall project risk score of 1.5 as shown in Table C-19, the overall risk level of the B8 Project is low.

Table C-19. The risk level of each metric in the B8 Project

Metric	Score	Risk Level
1. Number of personnel required for the development effort (people)	2	Medium
2. Defect density (defects / KLOC)	1	Low
3. Organizational complexity	2	Medium
4. Geographical distribution of teams	2	Medium
5. Requirements volatility	1	Low
6. Number of different job position types	1	Low
Overall risk level of the B8 Project	1.5	Low

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. CSE QUESTIONS

In this dissertation, we have provided answers to certain questions about measures of complexity in engineered complex systems.

1. Under what conditions does complexity arise?

Complexity can occur in time, space, and interactions. From this dissertation, we know that complexity arises when all of the seven features (nonlinearity, non-equilibrium, numerosity, diversity, interdependence, connectedness, and adaptivity) are present in the system. We also know that complexity arises because of intermediate values of these features. In the three software projects presented, we have shown that structural complexity, organizational complexity, temporal complexity, and technological complexity are four major sources that contribute to complexity in the Navy software projects. We can draw parallels from these four factors of complexity to 4 of the 7 building blocks of complexity—numerosity, connectedness, interdependence, and diversity.

2. What are the practical measures of complexity?

In this dissertation, we covered dozens of proposed measures of complexity from relevant literature. We discussed a few that have proven useful in any application, and none have been useful in every application. However, from the three software projects presented, the PCMM and associated methods to measure complexity demonstrated a practical approach to develop a complexity profile of an engineering project that consists of multiple complexity measures for project risk assessment. This could lead to a potential standardized complexity measurement for engineered systems.

3. What are the causes of emergence in a complex system?

Emergence is closely tied to complexity, nonlinearity, and constrained generating procedures (i.e., use of agents that interact with one another according to specific rules). We expect complex systems to exhibit emergence, and there are many paths leading to emergence. Thus, there are many types and "degrees" of emergence. As noted in the Chapter II literature review, potential emergent system elements include the following:

swarms, trails, task allocation and reallocation, mosaic images, crystallization, melting, film formation, connection generation, bridges, herding, driving, sorting, networks, synchronization, and distributed clustering [23]. Although we can use analogy to predict the results of similar systems, emergence is likely to pose many challenges to complex systems engineers.

4. What are the techniques to control complex systems?

To maintain control, complex systems require simplification or limits to their operational range. Some forms of prediction can also help to control complex systems. Fuzzy control, based on fuzzy logic, is another way to control complex systems. We may be able to impose control in many cases using these techniques, but such techniques will likely fail for some types of complex systems. We need additional research in complex systems control.

APPENDIX E. HEURISTIC APPROACHES TO REDUCING COMPLEXITY

In this dissertation, we sought to measure project complexity and suggest ways to reduce complexity. In this section, we list some heuristic approaches to reduce system complexity. Reducing complexity is not easy, nor is the process it requires. The following are heuristic approaches to reduce system and project complexity:

- Design modular hierarchical systems.
- Create many small assemblies with a reduced number of interfaces.
- Use distributed controls and processes when possible.
- Reduce the number of requirements and redundancies in the system.
- Simplify the design by reducing interdependence of components and use fewer types and kinds of parts.
- Use standard tools, procedures, and techniques.
- Enlist management and project sponsor support for project methodology.
- Involve users early in the project design cycle to minimize system requirement changes.
- Reduce the frequency of change in requirements, design environment, and system configuration.
- Improve clarity of communication.
- Use extensive modeling and simulation of SoI.
- Create prototypes and adapt proven designs.
- Learn from other industries.
- Train engineers so they become familiar with the system in development.
- Use social science principles to control the socio-political complexity.
- Implement Gaussian and power law analysis on a project problem.
- Reduce the geographical distances and time-zone discrepancies among team members.
- Change the problem definition or approach in such possible ways that reduces complexity of project activities.

- Combine and eliminate some reporting tasks and review activities.
- Clarify organizational roles and decision-making processes to best serve the project teams while also streamline on business processes and information systems.

LIST OF REFERENCES

[1] F. J. Dyson, "Is science mostly driven by ideas or by tools?" *Science Magazine*, vol. 338, no. 6113, pp. 1426–1427, December 14, Washington, DC, USA: AAAS, 2014. [Online]. Available: https://safe.menlosecurity.com/doc/docview/viewer/docNF8D7DD95FD3Bb1e00622456a532d69f9b231c2d3056d1d48104a74fce517c5cfc8f22b085b60

[2] C. N. Calvano and P. John, "Systems engineering in an age of complexity," *System Engineering*, vol. 7, no. 1, pp. 25–34, 2004. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=06021E51FF8187F1E3FE8001F419CE5B?doi=10.1.1.137.8013&rep=rep1&type=pdf

[3] M. V. Arena, O. Younossi, K. Brancato, I. Blickstein, and C. A. Grammich, "A macroscopic examination of the trends in U.S. military aircraft costs over the past several decades," RAND Corp. Report, 2008. [Online]. Available: http://www.rand.org/content/dam/rand/pubs/monographs/2008/RAND_MG696.pdf

[4] O.I. Olayode, L. K. Tartibu, and M. O. Okwu, "Application of artificial intelligence in traffic control system of non-autonomous vehicles at signalized road intersection," *Procedia CIRP Design Conference 2020*, pp. 194–200, vol. 91, August 18, New York, NY, USA: Elsevier Science Ltd., 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827120308076

[5] R. N. Charette, "This car runs on code," Feb. 1, New York, NY, USA: IEEE Spectrum, 2009. [Online]. Available: https://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

[6] T. Shale-Hester, "Driverless cars will require one billion lines of code, says JLR," Auto Express, April 16, Alfred Place, London: Dennis Publishing Ltd., 2019. [Online]. Available: https://www.autoexpress.co.uk/car-news/106617/driverless-cars-will-require-one-billion-lines-of-code-says-jlr

[7] J. Gertler, "F-35 Joint Strike Fighter (JSF) Program," RL30563, ver. 79, May 27, Washington, D.C., USA: Congressional Research Service, 2020. [Online]. Available: https://crsreports.congress.gov/product/pdf/RL/RL30563

[8] R. Harwood, "Meet the autonomous vehicle development 10x complexity challenge," *ANSYS Blog*, September 23, 2020. [Online]. Available: https://www.ansys.com/zh-cn/blog/meet-autonomous-vehicle-10x-complexity?

[9] M. Gottfredson and K. Aspinall, "Innovation versus complexity: what is too much of a good thing?" *Harvard Business Review*, Boston, MA, USA: Harvard Business Publishing, November 2005. [Online]. Available: https://hbr.org/2005/11/innovation-versus-complexity-what-is-too-much-of-a-good-thing?registration=success

[10] F. C. Mish (editor in chief), Webster's Ninth New Collegiate Dictionary, Merriam-Webster, Springfield, MA, USA, 1990.

[11] S. Wolfram, *A New Kind of Science*, Champaign, IL, USA: Wolfram Media, 2002

[12] E. Crawley, B. Cameron, and D. Selva, *System Architecture: Strategy and Product Development for Complex Systems*, First Edition, Boston, MA, USA: Pearson Education, 2016.

[13] S. E. Page, *Diversity and Complexity*, Princeton, NJ, USA: Princeton University Press, 2011

[14] D. Baccarini, "The concept of project complexity- a review," *International Journal of Project Management*, vol. 14, no. 4, pp. 201–204, New York, NY, USA: Elsevier Science Ltd., 1996. [Online]. Available: http://ieg.ifs.tuwien.ac.at/~aigner/projects/planninglines/evaluation/Project_Management/papers/baccarini96complexity.pdf

[15] L. A. Vidal and F. Marle, "Understanding project complexity: implications on project management," *Kybernetes*, vol. 37, no. 8, pp. 1094–1110, Bingley, UK: Emerald Publishing Ltd., September 2008. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01215364/document

[16] K. Remington, R. Zolin and R. Turner, "A model of project complexity: distinguishing dimensions of complexity from severity," *Proceedings of the 9th International Research Network of Project Management Conference*, October 11–13, Berlin, Germany, 2009. [Online]. Available: https://eprints.qut.edu.au/29011/1/c29011.pdf

[17] B. P. Dao, "Exploring and measuring project complexity," Ph.D. Dissertation, Texas A&M University, College Station, TX, USA, August 2016. [Online]. Available: https://oaktrust.library.tamu.edu/bitstream/handle/1969.1/158012/DAO-DISSERTATION-2016.pdf?sequence=1&isAllowed=y

[18] S. Page, "Understanding complexity," The Great Courses, video lecture series available from The Teaching Company, Chantilly, VA, USA, 2009.

[19] E. Chaisson, *Cosmic Evolution: The Rise of Complexity in Nature*, Cambridge, MA, USA: Harvard University Press, 2001.

[20] DARPA, "Breakthrough technologies for national security," March 2015. [Online]. Available: https://www.darpa.mil/attachments/DARPA2015.pdf

[21] DARPA, "Adaptive vehicle make (AVM)," 2014. [Online]. Available: https://www.darpa.mil/program/adaptive-vehicle-make

[22] O. L. de Weck, "Fundamental of systems engineering," Massachusetts Institute of Technology: MIT Open Courseware 16.842, MIT, Massachusetts, USA, Dec. 2015 [Online]. Available: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-842-fundamentals-of-systems-engineering-fall-2015/lecture-notes/MTI16_842F15_Ses12_FutofSE.pdf

[23] M. R. Blackburn and S. Ray, "Reducing verification costs through practical formal methods: A survey," Stevens Institute of Technology, Hoboken, New Jersey, USA, Dec. 2010. [Online]. Available: https://www.researchgate.net/publication/303676169_Reducing_Verification_Costs_through_Practical_Formal_Methods_A_Survey

[24] R. Valerdi, "Constructive systems engineering cost model (COSYSMO)," Ph.D. Dissertation, University of Southern California, Los Angeles, USA, August 2005. [Online]. Available: https://www.researchgate.net/publication/242452004_The_constructive_systems_engineering_cost_model_COSYSMO

[25] R. Schmidt, K. Lyytinen, M. Keil, and P. Cule, "Identifying software project risks: an international Delphi study," *Journal of Management Information Systems,* vol. 17, no. 4, pp. 5–36, Philadelphia, PA, USA: Taylor & Francis Group, 2001. [Online]. Available: https://www.researchgate.net/profile/Kalle_Lyytinen/publication/220591356_Identifying_Software_Project_Risks_An_International_Delphi_Study/links/00b7d517ae60719bc2000000/Identifying-Software-Project-Risks-An-International-Delphi-Study.pdf

[26] L. Wallace, M. Keil and A. Rai, "Understanding software project risk: a cluster analysis," *Information & Management*, vol. 42, pp. 115–125, New York, NY, USA: Elsevier Science Ltd., March 2004. [Online]. Available: http://www-public.imtbs-tsp.eu/~gibson/Teaching/Teaching-ReadingMaterial/WallaceKeilRai04.pdf

[27] R. Giachetti, *Design of Enterprise Systems: Theory, Architecture, and Methods*, CRC Press, Boca Raton, FL, USA: Taylor and Francis Group, LLC, 2010.

[28] R. E. Giachetti, "Understanding interdependence in enterprise systems: A model and measurement formalism," Department of Industrial & Systems Engineering, Florida International University, Miami, Florida, USA, September 2006. [Online]. Available: https://www.researchgate.net/publication/225229234_Understanding_Interdependence_in_Enterprise_Systems_A_Model_and_Measurement_Formalism

[29] R. C. Harney, "Introduction to complexity and complex systems engineering," class notes for Complex Systems Engineering, Department of Systems Engineering, Naval Postgraduate School, Monterey, CA, USA, fall 2017.

[30] A. M. Rustad, K. G. Hanssen, "Machine learning in control systems: an overview of the state of the art," *Artificial Intelligence XXXV*, pp. 250–265, *38th SGAI International Conference on Artificial Intelligence*, December 11–13, Cambridge, UK, 2018

[31] The Mathworks website. "What is deep learning?" [Online]. Available: https://www.mathworks.com/discovery/deep-learning.html#:~:text=Deep%20learning%20is%20a%20machine,a%20pedestrian%20from%20a%20lamppost.

[32] Wikipedia. "Likert scale." [Online]. Available: https://en.wikipedia.org/wiki/Likert_scale

[33] N. Subramanian and L. Chung, "Metrics for software adaptivity," Department of Computer Science, University of Texas, Dallas, Texas, USA. [Online]. Available: https://personal.utdallas.edu/~chung/ftp/sqm.pdf

[34] CANES, "What are non-equilibrium systems," King's College, London, UK. [Online]. Available: https://www.kcl.ac.uk/noneqsys/about-us/what-are-non-equilibrium-systems

[35] J. B. Clark and D. R. Jacques, "Practical measurement of complexity in dynamic systems," *New Challenges in Systems Engineering and Architecting Conference on Systems Engineering Research* (CSER), St. Louis Missouri, USA, March 2012. [Online]. Available: https://www.researchgate.net/publication/257719321_Practical_measurement_of_complexity_in_dynamic_systems

[36] Investopedia. "Definition of nonlinearity." [Online]. Available: https://www.investopedia.com/terms/n/nonlinearity.asp

[37] Wikipedia. "Definition of nonlinear systems." [Online]. Available: https://en.wikipedia.org/wiki/Nonlinear_system

[38] S. H. Strogatz, *Nonlinear Dynamics and Chaos*, Second Edition, Boulder, CO, USA: Westview Press, 2015.

[39] H. Lee, V. Padmanabhan, and S. Whang, "Information distortion in a supply chain: The Bullwhip effect," Management Science, InformsPubsOnline, Catonsville, MD, USA, April 1997. [Online]. Available: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.43.4.546

[40] S. A. Sheard, "Assessing the impact of complexity attributes on system development project outcomes," p. 10, Ph.D. Dissertation, Stevens Institute of Technology, Hoboken, NJ, USA, 2012.

[41] L. Hoogduin, "A review of common characteristics of complex systems," *Decision Making in a Complex and Uncertain World*, an online course, University of Groningen," Groningen, Netherlands, 2020. [Online]. Available: https://www.futurelearn.com/courses/complexity-and-uncertainty/0/steps/1836

[42] CAGW Staff, "California's $100 billion nightmare high-speed rail project," Washington, DC, USA: CAGW, July 1, 2020. [Online]. Available: https://www.cagw.org/thewastewatcher/californias-100-billion-nightmare-high-speed-rail-project

[43] R. Pallardy, "Deepwater horizon oil spill," Chicago, IL, USA: Britannica Group, July 9, 2010. [Online]. Available: https://www.britannica.com/event/Deepwater-Horizon-oil-spill

[44] P. Checkland, *Systems Thinking, Systems Practice*, Chichester, UK: J. Wiley, 1981.

[45] J. H. Holland, *Emergence: From Chaos to Orders*, New York, NY, USA: Oxford University Press, 2000.

[46] J. H. Miller and S. E. Page, *Complex Adaptive Systems*, Princeton, NJ, USA: Princeton University Press, 2007, p. 3, p. 35 & pp. 40–43.

[47] Dictionary. "Definition of limited predictability." [Online]. Available: https://www.dictionary.com/browse/predictability

[48] R. E. A. van Emmerik, S. W. Ducharme, A. C. Amado, and J. Hamill, "Comparing dynamical systems concepts and techniques for biomechanical analysis," *Journal of Sport and Health Science*, vol. 5, issue 1, pp. 3–13, New York, NY, USA: Elsevier Science Ltd., March 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095254616000156

[49] Magazine of the University of Utah, "A New Hope." [Online]. Available: https://magazine.utah.edu/issues/winter-2020/a-new-hope/

[50] F. Heylighen, edited by M. Bates and M. N. Maack, "Complexity and self-organization," Free University of Brussels in Belgium, Philadelphia, PA, USA: Taylor & Francis Group, 2008. [Online]. Available: http://pespmc1.vub.ac.be/Papers/ELIS-complexity.pdf

[51] N. Raichman, T. Gabay, Y. Katsir, Y. Shapira, and E. Ben-Jacob, "Engineered self-organization in natural and man-made systems," School of Physics and Astronomy, Tel-Aviv University, Tel-Aviv, Israel: January 2004. [Online]. Available: https://www.academia.edu/12230502/Engineered_Self-Organization_In_Natural_and_Man-Made_Systems

[52] P. Bak, *How Nature Works: The Science of Self-Organized Criticality*, New York, NY, USA: Copernicus Press, 1996.

[53] Wikipedia. "Definition of uncertainty." [Online]. Available: https://en.wikipedia.org/wiki/Uncertainty

[54] N. P. Suh, *Axiomatic Design: Advances and Applications*, New York, NY, USA: Oxford University Press, 2001.

[55] M. Bordo and A. Redish, "Putting the 'system' in the international monetary system," VOX CEPR policy portal, June 2013. [Online]. Available: https://voxeu.org/article/putting-system-international-monetary-system

[56] Infoplease. "International Finance: The international monetary system." [Online]. Available: https://www.infoplease.com/homework-help/social-studies/international-finance-international-monetary-system

[57] J. R. San Cristóbal, L. Carral, E. Diaz, J. A. Fraguela, and G. Iglesias, "Complexity and project management: a general overview," vol. 2018, London, UK: Hindawi Ltd., October 2018. [Online]. Available: https://www.hindawi.com/journals/complexity/2018/4891286/

[58] INCOSE Complex Systems Working Group, "A complexity primer for systems engineers," INCOSE, July 2015. [Online]. Available: https://www.aiaa.org/uploadedFiles/Events/Complexity%20Primer%20for%20SE%20July%202015.pdf

[59] T. Healy, "The unanticipated consequences of technology," Santa Clara University, CA, USA: Markkula Center for Applied Ethics, April 6, 2005. [Online]. Available: https://www.scu.edu/ethics/focus-areas/technology-ethics/resources/the-unanticipated-consequences-of-technology/

[60] J. Horgan, "From complexity to perplexity," *Scientific American*, June 1995. [Online]. Available: http://www2.econ.iastate.edu/tesfatsi/hogan.complexperplex.html

[61] O. Maimon and D. Braha, "Communications: a proof of the complexity of design," *Kybernetes*, vol. 21, no. 7, pp. 59–62, Bingley, UK: Emerald Publishing Ltd., July 1992.

[62] D. Braha and O. Maimon, "On the complexity of the design synthesis problem," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, no. 1, 1996.

[63] D. Braha and Y. Bar-Yam, "The structure and dynamics of complex product design in complex engineered systems: A new paradigm in complex engineered systems," Cambridge, MA, USA: Springer, NECSI, 2006, pp. 40–71.

[64] J. R.A. Maier and G. M. Fadel, "Understanding the complexity of design in complex engineered systems: A new paradigm in complex engineered systems," Cambridge, MA, USA: Springer, NECSI, 2006, pp. 122–140.

[65] S. A. Sheard and D. A. Mostashari, "A complexity typology for systems engineering," Hoboken, NJ, USA: Stevens Institute of Technology, 2010. [Online]. Available: https://www.researchgate.net/publication/228741550_A_Complexity_Typology_for_Systems_Engineering

[66] R. Stevens, "Engineering enterprise systems: challenges and prospects," McLean, VA, USA: The MITRE Corporation, 2006. [Online]. Available: https://www.mitre.org/sites/default/files/pdf/06_0342.pdf

[67] S. D. Eppinger, D. E. Whitney, R. P. Smith, and D. A. Gebala, "A model-based method for organizing tasks in product development," MIT, *Research in Engineering Design*, vol. 6, issue 1, pp. 1–13, New York, NY, USA: Springer March 1994. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/2468/swp-3569-29894176.pdf?sequence=1

[68] S. H. Strogatz, *Nonlinear Dynamics and Chaos*, Second Edition, Boulder, CO, USA: Westview Press, 2015.

[69] M. E. J. Newman, *Networks: An Introduction*, Oxford, UK: Oxford University Press, 2010.

[70] A. L. Barabási, *Linked*, New York, NY, USA: Penguin Books, 2003.

[71] P. Erdős, and A. Renyi, "On random graphs," *Publicationes Mathematicae.* vol. 6, pp. 290–297, University of Debrecen, Debrecen, Hungary, 1959.

[72] B. Bollobas, *Random Graphs*, second edition, New York, NY, USA: Cambridge University Press, 2001

[73] M. J. Jeger, M. Pautasso, O. Holdenrieder, M. W. Shaw, "Modelling disease spread and control in networks: implications for plant sciences," *New Phytologist*, vol. 174, issue 2, pp. 279–297, April 2007. [Online]. Available: https://nph.onlinelibrary.wiley.com/doi/full/10.1111/j.1469-8137.2007.02028.x

[74] D. J. Watts, "Small worlds," Princeton, NJ, USA: Princeton University Press, 2003.

[75] D. J. Watts and S. H. Strogatz, "Collective dynamics of small world networks," *Nature*, vol. 393, p. 440, June 1998.

[76] G. Caldarelli, *Scaled-Free Networks*, Oxford, UK: Oxford University Press, 2007

[77] M. Faloutsos, P. Faloutsos, P. and C. Faloutsos, "On power law relationships of the internet topology," *Computer Communications Review*, vol. 29, no. 251, 1999.

[78] W. Weaver, "Science and complexity," *American Scientist*, vol. 36, pp. 536–544, Research Triangle Park, NC, USA: Sigma Xi, The Scientific Research Honor Society, 1948. [Online]. Available: http://people.physics.anu.edu.au/~tas110/Teaching/Lectures/L1/Material/WEAVER1947.pdf

[79] A. A. Minai, D. Braha and Y. Bar-Yam, "Complex engineered systems: a new paradigm in complex engineered systems," Cambridge, MA, USA: Springer, NECSI, pp. 1–21, pp. 224–232 & pp. 234–235, 2006.

[80] A. Clauset, "Inference, models and simulation for complex systems," lecture notes, Computer Science Department, Santa Fe Institute, Santa Fe, NM, USA, summer 2011. [Online]. Available: http://tuvalu.santafe.edu/~aaronc/courses/7000/csci7000-001_2011_L2.pdf

[81] M. Mitchell, *Complexity: A Guided Tour,* New York, NY, USA: Oxford University Press, 2009.

[82] C. Weber, "What is 'complexity'?" *Proceedings of the 15th International Conference on Engineering Design (ICED 05),* August 15–18, Melbourne, Australia, 2005. [Online]. Available: https://www.designsociety.org/publication/23103/WHAT+IS+%E2%80%9CCOMPLEXITY%E2%80%9D%3F

[83] T. M. Williams, "The need for new paradigms for complex projects," *International Journal of Project Management*, vol. 17, no. 5, pp. 269–273, New York, NY, USA: Elsevier Science Ltd and IPMA, 1999. [Online]. Available: http://ieg.ifs.tuwien.ac.at/~aigner/projects/planninglines/evaluation/Project_Management/papers/williams99complexity.pdf

[84] K. Remington and J. Pollack, *Tools for Complex Projects*, Oxford, UK: Routledge, February 2008.

[85] G. Lee, W. Xia, "Development of a measure to assess the complexity of information systems development projects," Twenty-Third International Conference on Information Systems, 2002. http://ieg.ifs.tuwien.ac.at/~aigner/projects/planninglines/evaluation/Project_Management/papers/0211_100102.pdf Accessed on June 21, 2019.

[86] B. P. Dao, "Exploring and measuring project complexity," Ph.D. Dissertation, Texas A&M University, August 2016. https://oaktrust.library.tamu.edu/bitstream/handle/1969.1/158012/DAO-DISSERTATION-2016.pdf?sequence=1&isAllowed=y Accessed on June 21, 2019.

[87] M. E. Conway, "How do committees invent?" Datamation, Nashville, TN, USA: TechnologyAdvice, April 1968. [Online]. Available: https://www.melconway.com/Home/pdf/committees.pdf

[88] R. J. Heaslip, "Managing organizational complexity: how to optimize the governance of programs and projects to improve decision making," PMI White Papers, April 2015. [Online]. Available: https://www.pmi.org/learning/library/managing-organizational-complexity-11139

[89] O. U. Ugbomhe and A. B. Dirisu, "Organizational structure: dimensions, determinants and managerial implication," International Journal of Economic Development Research and Investment, vol. 2, no. 2, Ikot Ekpene, Akwa Ibom State, Nigeria: International Centre for Integrated Development Research, August 2011. [Online]. Available: http://www.icidr.org/ijedri_vol2no2_august2011/Organizational%20Structure%20Dimensions,%20Determinants%20and%20Managerial%20Implication.pdf

[90] S. Lloyd, "Measures of complexity: A non-exhaustive list," *IEEE Control Systems Magazine,* August 2001. [Online]. Available: http://csc.ucdavis.edu/~cmg/Group/readings/SL-MeasOfComplexity.pdf

[91] W. Weaver and C. E. Shannon, "Recent contributions to the mathematical theory of communication," pp. 4–9, September 1949. [Online]. Available: https://pdfs.semanticscholar.org/c4ee/686f5dd14ac83c4b10a8bce9a62341ea0a3a.pdf

[92] M. Gell-Mann and S. Lloyd, "Information measures, effective complexity, and total information," *Complexity*, vol. 2, issue 1, pp. 44–52, September 1996.

[93] B.M. Arteta and R. E. Giachetti, "A measure of agility as the complexity of the enterprise system," *Robotics and Computer-Integrated Manufacturing*, New York, NY, USA: Elsevier Science Ltd., 2004.

[94] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems of Information Transmission,* vol. 1, no. 1, pp. 3–11, 1965. [Online]. Available: https://www.tandfonline.com/doi/pdf/10.1080/00207166808803030

[95] DARPA "Historical schedule trends with complexity." [Online]. Available: https://media.springernature.com/original/springer-static/image/chp:10.1007%2F978-3-319-38756-7_9/MediaObjects/327865_1_En_9_Fig10_HTML.gif

[96] IFPUG. "A. Albrecht of IBM, function point analysis." [Online]. Available: https://www.ifpug.org/about-function-point-analysis/

[97] International Function Point Users Group (IFPUG), "Function point counting practices manual 4.1," 1999. [Online]. Available: http://csevgi.bilkent.edu.tr/courses/ctis359/IFPUG_Counting_Practices_Manual_4.1.pdf

[98] Parvathy, "Function point analysis – introduction and fundamentals," January 09, New York, NY, USA: Fingent, 2020. [Online]. Available: https://www.fingent.com/blog/function-point-analysis-introduction-and-fundamentals/

[99] K. Sinha, "Structural complexity and its implications for design of cyber-physical systems," Ph.D. Dissertation, Massachusetts Institute of Technology, Massachusetts, USA, 2014.

[100] H. A. Simon, "The architecture of complexity," *Proceedings of the American Philosophical Society*, vol. 106, issue 6, pp. 467–482, December 1962. [Online]. Available: http://www2.econ.iastate.edu/tesfatsi/ArchitectureOfComplexity.HSimon1962.pdf

[101] J. E. McCann and D. L. Ferry, "An approach to assessing and managing inter-unit interdependence," *Academy of Management Review*, vol. 4, pp. 113–119, 1979.

[102] B. Victor and R. S. Blackburn, "Interdependence: an alternative conceptualization," *Academy of Management Review,* vol. 12, pp. 486–498, 1987. [Online]. Available: https://www.researchgate.net/publication/242546051_Interdependence_An_Alternative_Conceptualization

[103] S. Lloyd and H. Pagels, "Complexity as thermodynamic depth," *Annals of Physics*, vol. 188, pp. 186–213, November 1988.

[104] C. Perrow, *Normal Accidents: Living with High-Risk Technologies*, Princeton, NJ, USA: Princeton University Press, 1999, pp. 62–100.

[105] R. Lopez-Ruiz, H. Mancini, and X. Calbet, "A statistical measure of complexity," Physics Letters A, vol. 209, pp. 321–326, 1995. [Online]. Available: https://arxiv.org/pdf/1009.1498.pdf

[106] SEJ, "What are Google algorithms?" Boca Raton, FL, USA: Search Engine Journal, June 15, 2021. [Online]. Available: https://www.searchenginejournal.com/google-algorithm-history/

[107] Wikipedia. "PageRank." [Online]. Available: https://en.wikipedia.org/wiki/PageRank

[108] A. Thevenot, "Particle swarm optimization (PSO) visually explained," Towards Data Science, Dec. 21, 2020. [Online]. Available: https://towardsdatascience.com/particle-swarm-optimization-visually-explained-46289eeb2e14

[109] M. G. Alonso and P. Duysinx, "Particle swarm optimization (PSO): an alternative method for composite optimization," 10th World Congress on Structural and Multidisciplinary Optimization, May 19–24, 2013, Orlando, FL, USA. [Online]. Available: https://mae.ufl.edu/mdo/Papers/5334.pdf

[110] C. H. Bennett, "Logical depth and physical complexity," in *The Universal Turing Machine: A Half-Century Survey* by R. Herken, New York, NY, USA: Oxford University Press, 1988, pp. 227–257. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.4331&rep=rep1&type=pdf

[111] F. Allgöwer, "Definition and computation of a nonlinearity measure," IFAC Nonlinear Control Systems Design, Tahoe City, CA, USA, 1995. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017468406

[112] Wikipedia. "Definition of multimodality." [Online]. Available: https://en.wikipedia.org/wiki/Multimodality

[113] A. V. Deshmukh, J. J. Talavage, and M. M. Barash, "Complexity in manufacturing systems. Part 1: analysis of static complexity," IIE Transactions, pp. 645–655, July 1998.

[114] G. Frizelle and E. Woodcock, "Measuring complexity as an aid to develop operational strategy," *International Journal of Operation & Production Management*, vol. 15, no. 5, pp. 26–39, May 1995.

[115] S. Sivadasan, J. Efstathiou, G. Frizelle, R. Shirazi, and A. Calinescu, "An information-theoretic methodology for measuring the operational complexity of supplier–customer systems," *International Journal of Operation & Production Management*, vol. 22, no. 1, pp. 80–102, 2002.

[116] M. Meyer and A. Lehnerd, *The Power of Product Platforms*, Florence, MA, USA: Free Press, March 1997.

[117] L. Minati et al., "Connectivity influences on nonlinear dynamics in weakly-synchronized networks: insights from Rössler systems, electronic chaotic oscillators, model and biological neurons," *IEEE Access*, vol. 7, pp. 174793-174821, New York, NY, USA: IEEE, December 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8918231/

[118] M. Bailey, R. Cao, T. Kuchler, J. Stroebel, and A. Wong, "Social connectedness: measurement, determinants, and effects," *Journal of Economic Perspective*s, vol. 32, number 3, pp. 259–280, 2018. [Online]. Available: http://pages.stern.nyu.edu/~jstroebe/PDF/JEP_SCI.pdf

[119] T. J. Allen, "Managing the flow of technology: technology transfer and the dissemination of technological information within the R&D organization," Cambridge, MA, USA: MIT Press, 1984.

[120] M. Greenwood-Nimmo, V. H. Nguyen, and Y. Shin, "Measuring connectedness of the global economy," Department of Economics, University of Melbourne, Melbourne, Australia, January 2015. [Online]. Available: http://www.greenwoodeconomics.com/(MG4)GNS_GC.pdf

[121] M. M. Danziger, A. Bashan, Y. Berezin, L. Shekhtman, and S. Havlin, "An introduction to interdependent networks," International Conference on Nonlinear Dynamics of Electronic Systems, *Communications in Computer and Information Science*, vol 438. pp. 189–202, Cham, Switzerland: Springer International Publishing, July 2014. [Online]. Available: https://www.researchgate.net/publication/278712957_An_Introduction_to_Interdependent_Networks

[122] J. F. Amaral J. Dias, and J. C. Lopes, "Complexity as interrelatedness: an input-output approach," EcoMod/IIOA Conference: Input-Output and General Equilibrium (Data, Modeling and Policy Analysis) September 2–4, Brussels, Belgium, 2004. [Online]. Available: https://www.researchgate.net/publication/228430598_Complexity_as_Interrelatedness_an_Input-Output_Approach

[123] J. C. Domercant, and D. N. Maviris, "Measuring the architectural complexity of military systems-of-systems," Aerospace Systems Design Laboratory, Georgia Institute of Technology, Atlanta, GA, USA, Dec. 2010. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/stamp/stamp.jsp?tp=&arnumber=5747653

[124] W. R. Ashby, "An introduction to cybernetics," *Second Impression*, London, UK: Chapman & Hall Ltd., 1957. [Online]. Available: http://pespmc1.vub.ac.be/books/IntroCyb.pdf

[125] T. Broekel, "Using structural diversity to measure the complexity of technologies," PLOS One Journal, San Francisco, CA, USA: PLOS, May 2019. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0216856

[126] S. Chan, "Complex adaptive systems," Research Seminar in Engineering Systems, November 2001. [Online]. Available: http://web.mit.edu/esd.83/www/notebook/Complex%20Adaptive%20Systems.pdf

[127] A. Siddiqa, M. A. Shah, H. A. Khattak, A. Akhunzada, I. Ali, Z. B. Razak, and A. Gani, "Social internet of vehicles: complexity, adaptivity, issues and beyond," IEEE Access, New York, NY, USA: IEEE, Nov. 2018. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/stamp/stamp.jsp?tp=&arnumber=8478269

[128] M. Tamersoy, E. E. Ekinci, R. C. Erdur, and O. Dikenelli, "A requirements model for adaptive multi-organizational systems," *11th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Washington, DC, USA: IEEE Computer Society, 2017. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/stamp/stamp.jsp?tp=&arnumber=8064028

[129] "Defense acquisitions: Stronger management practices are needed to improve DOD's software-intensive weapon acquisitions," Highlights of GAO-04-393, a report to the Committee on Armed Services, U.S. Senate, March 2004. [Online]. Available: https://www.govinfo.gov/content/pkg/GAOREPORTS-GAO-04-393/html/GAOREPORTS-GAO-04-393.htm

[130] "Weapon systems annual assessment: Limited use of knowledge-based practices continues to undercut DOD's investments," GAO-19-336SP, a report to Congressional Committees, May 2019. [Online]. Available: https://www.gao.gov/assets/700/698933.pdf

[131] K. B. Hass and L. B. Lindbergh, "The bottom line on project complexity: applying a new complexity model," PMI Global Congress, Newtown Square, PA, USA: Project Management Institute, October 2010. [Online]. Available: https://www.pmi.org/learning/library/project-complexity-model-competency-standard-6586

[132] E. Oviedo, "Control flow, data flow and program complexity," *Proceedings of the IEEE COMPSAC*, pp. 146–152, 1980.

[133] J. K. Kearney, R. L. Sedlmeyer, W. B. Thompson, M. A. Gray, and M.A. Adler, "Software complexity measurement," vol. 29, no. 11, Communication of the ACM, New York, NY, USA, November 1986. [Online]. Available: http://sunnyday.mit.edu/16.355/kearney.pdf

[134] Wikipedia. “Sources line of code.” [Online]. Available: https://en.wikipedia.org/wiki/Source_lines_of_code

[135] N. Arshad, “Software metrics,” LUMS, Lahore, Pakistan. [Online]. Available: http://suraj.lums.edu.pk/~cs563/metrics.pdf

[136] A. H. Watson and T. J. McCabe, “Structured testing: a testing methodology using the cyclomatic complexity metric,” p. 10, pp. 15–33, p. 96, September 1996. [Online]. Available: http://mccabe.com/pdf/mccabe-nist235r.pdf

[137] Sourceforge. “Eclipse metrics plugin.” [Online]. Available: http://eclipse-metrics.sourceforge.net/

[138] Github. “ReSharper plugin.” [Online]. Available: https://github.com/JetBrains/resharper-cyclomatic-complexity

[139] M. H. Halstead, *Elements of Software Science*, Rochester, NY, USA: Elsevier North-Holland, 1977.

[140] McCabe. “McCabe software tools.” [Online]. Available: http://www.mccabe.com/pdf/McCabe%20IQ%20Metrics.pdf

[141] S. Henry and D. Kafura, *The evaluation of software systems’ structure using quantitative software metrics*, *software: Practice and experience*, vol. 14, issue 6, pp. 561–573, John Wiley & Sons, June 1984. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380140606

[142] S. Henry, “Information flow metrics for the evaluation of operating systems’ structure,” pp. 64–66, pp. 92–94, PhD. Dissertation, Iowa State University, Ames, IA, USA, 1979. [Online]. Available: https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=8212&context=rtd

[143] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” Sloan School of Management, MIT, Massachusetts, USA, pp. 12–27, December 1993. [Online]. Available: http://www.eso.org/~tcsmgr/oowg-forum/TechMeetings/Articles/OOMetrics.pdf

[144] R. Shatnawi and W. Li, “The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process,” *Journal of Systems and Software*, vol. 81, no. 11, pp. 1868–1882, January 2008. [Online]. Available: https://www.academia.edu/29908210/The_effectiveness_of_software_metrics_in_identifying_error-prone_classes_in_post-release_software_evolution_process

[145] H. A. Sahraoui, R. Godin and T. Miceli, “Can metrics help to bridge the gap between the improvement of OO design quality and its automation?” *Proceedings of 16th IEEE International Conference on Software Maintenance*, 2000. [Online]. Available: https://www.iro.umontreal.ca/~sahraouh/papers/ICSM00.pdf

[146] E. E. Ogheneovo, "On the relationship between software complexity and maintenance costs," *Journal of Computer and Communications,* vol. 2, no. 14, Nov. 21, 2014. [Online]. Available: https://www.semanticscholar.org/paper/On-the-Relationship-between-Software-Complexity-and-Ogheneovo/84a809cb18443457eb91e175f3bc42fa240c4d4f

[147] Y. Shi and S. Xu, "A new method for measurement and reduction of software complexity," *Tsinghua Science and Technology*, vol. 12, no. S1, pp. 212-216, July 2007, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/document/6074053

[148] T. A. Khalid, E. Yeoh, “Early cost estimation of software reworks using fuzzy requirement-based model,” *2017 International Conference on Communication, Control, Computing and Electronic Engineering* (ICCCCEE), January 16–18, 2017.

[149] R. C. Martin, *Agile Development: Principles, Patterns, and Practices*, Upper Saddle River, NJ, USA: Prentice Hall, 2003

[150] N. Ahmad and P. A. Laplante, “Employing expert opinion and software metrics for reasoning about software,” *Third IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pp. 119–124, Columbia, MD, USA, September 2007. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4351396

[151] W. C. Josey and K. England, “Utilizing a project profile matrix to determine project management requirements,” PMI Global Congress 2009, Newtown Square, PA, USA: Project Management Institute, Oct. 2009. [Online]. Available: https://www.pmi.org/learning/library/utilizing-project-profile-matrix-pm-requirements-10598

[152] M. Deery, “How much will I earn as a full-stack developer in 2021?” April 30, 2021. [Online]. Available: https://careerfoundry.com/en/blog/web-development/full-stack-developer-salary-guide/

[153] “The true cost of hiring developers in the U.S., the UK, Germany, the Netherlands,” Daxx, April 1, 2021. [Online]. Available: https://www.daxx.com/blog/development-trends/how-much-costs-hire-developer

[154] S. M. Rowlinson, “An analysis of factors affecting project performance in industrial building,” Ph.D. Dissertation, Brunel University, London, UK, 1988. [Online]. Available: https://www.academia.edu/25339950/AN_ANALYSIS_OF_FACTORS_AFFECTING_PROJECT_PERFORMANCE_IN_INDUSTRIAL_BUILDING_with_particular_reference_to_Design_Build_contracts

[155] G. Stark, P. Oman, A. Skillicorn, and R. Ameele, “An examination of the effects of requirements changes on software maintenance releases,” *Journal of Software Maintenance Research and Practice*, vol. 11, pp. 293–309, Hoboken, New jersey, USA: John Wiley & Sons, 1999. [Online]. Available: https://www.researchgate.net/publication/220674128_An_Examination_of_the_Effects_of_Requirements_Changes_on_Software_Maintenance_Releases

[156] C. Brockmann and G. Girmscheid, “Complexity of megaprojects,” Conference paper, May 14–18, CIB World Building Congress, Cape Town, South Africa, 2007. [Online]. Available: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/4495/eth-845-01.pdf

[157] H. Wood and P. Ashton, “The factors of project complexity,” pp. 69–80, 18th CIB World Building Congress, Salford, UK: CIB, Jan. 2010. [Online]. Available: https://www.irbnet.de/daten/iconda/CIB_DC24048.pdf

[158] V. A. Schwandt, “Measuring organizational complexity and its impact on organizational performance – a comprehensive conceptual model and empirical study,” Ph.D. Dissertation, University of Berlin, Germany, July 2009. [Online]. Available: https://d-nb.info/1000048349/34

[159] Y. K. Malaiya and J. Denton, “Requirements volatility and defect density,” *Proceedings 10th International Symposium on Software Reliability Engineering* (Cat. No. PR00443), pp. 285–294, Boca Raton, FL, USA: IEEE, 1999. [Online]. Available: https://www.cs.colostate.edu/~malaiya/reqvol.pdf

[160] A. M. Neufelder, “Current defect density statistics,” Bluffton, SC, USA: SoftRel, LLC, 2007. [Online]. Available: http://www.softrel.com/Current%20defect%20density%20statistics.pdf

[161] D. Pfahl and K. Lebsanft, “Using simulation to analyze the impact of software requirement volatility on project performance,” *Information and Software Technology*, vol. 42, issue 14, pp. 1001–1008, New York, NY, USA: Elsevier Science Ltd., November 2000. [Online]. Available: https://www.researchgate.net/publication/221219179_Using_Simulation_to_Visualise_and_Analyse_Product-Process_Dependencies_in_Software_Development_Projects

[162] H. Parsons-Hann and K. Liu, "Measuring requirements complexity to increase the probability of project success," *Proceedings of the Seventh International Conference on Enterprise Information Systems*, vol. 3, pp. 434–438, Miami, USA, 2005. [Online]. Available: https://www.scitepress.org/Papers/2005/25481/25481.pdf

[163] S. W. Azim, "Understanding and managing project complexity," Ph.D. Thesis, University of Manchester, Manchester, UK, 2010. [Online]. Available: https://www.escholar.manchester.ac.uk/api/datastream?publicationPid=uk-ac-man-scw:121030&datastreamId=FULL-TEXT.PDF

[164] B. Waber, J. Magnolfi, and G. Lindsay, "Workspaces that move people," Harvard Business Review, pp. 69–77, Boston, MA, USA: Harvard Business Publishing, Oct. 2014, [Online]. Available: https://hbr.org/2014/10/workspaces-that-move-people

[165] Project Smart, 2014. "The Standish Group report CHAOS." [Online]. Available: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

[166] N. Nurmuliania, D. Zowghi and S. Fowell, "Analysis of requirements volatility during software development life cycle," AWRE, Adelaide, Australia, 2006. [Online]. Available: http://www.robertfeldt.net/courses/reqeng/papers/nurmulian_2004_analysis_of_req_volatility.pdf

[167] B. Sharif, S. A. Khan, and M. W. Bhatti, "Measuring the impact of changing requirements on schedule variance: an empirical investigation," *2012 International Conference on Software and Computer Applications*, Singapore, 2012. [Online]. Available: https://studylib.net/doc/13136319/measuring-the-impact-of-changing-requirements-on-schedule

[168] H. Dev and R. Awasthi, "A systematic study of requirement volatility during software development process," *IJCSI International Journal of Computer Science Issues*, vol. 9, issue 2, no 1, March 2012. [Online]. Available: http://ijcsi.org/papers/IJCSI-9-2-1-527-533.pdf

[169] M. Peña and R. Valerdi, "Characterizing the impact of requirements volatility on systems engineering effort," 25th Annual COCOMO Forum, Los Angeles, CA, USA: University of Southern California, November 2010. [Online]. Available: http://dspace.mit.edu/bitstream/handle/1721.1/84023/CPS_101104_Pena_Valerdi_COCOMO.pdf.pdf;sequence=1

[170] J. M. Feland III and L. J. Leifer, "Requirement volatility metrics as an assessment instrument for design team performance prediction," The International Journal of Engineering Education, vol. 17, no. 4–5, pp. 489–492, Chicago, IL, USA: Templus Publications, 2001. [Online]. Available: https://www.ijee.ie/articles/Vol17-4and5/IJEE1238.pdf

[171] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” *27th International Conference on Software Engineering*, May 15–21, St. Louis, Missouri, USA: ACM, 2005. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/icse05churn.pdf

[172] W. V. Heydebrand, “Comparative organizations: the results of empirical inquiry,” Englewood Cliffs, NJ, USA: Prentice-Hall Inc., 1973.

[173] J. D. Caprace and P. Rigo, “A complexity metric for practical ship design,” *11th International Symposium on Practical Design of Ships and Other Floating Structures*, vol. 1, Brazil, 2010. [Online]. Available: https://www.researchgate.net/publication/258835031_A_Complexity_Metric_for_Practical_Ship_Design

[174] P. Leukert, “IT complexity: model, measure and master,” *Capco Thoughts white paper*, New York, NY, USA: Capco, July 27, 2011. [Online]. Available: https://www.slideshare.net/CapcoGlobal/it-complexity-model-measure-and-master

[175] J. C. Domerçant and D. N. Mavris, "Measuring the architectural complexity of military Systems-of-Systems," 2011 Aerospace Conference, 2011, pp. 1-16, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore.ieee.org/document/5747653

[176] P. Malone and L. Wolfarth, "Measuring system complexity to support development cost estimates," 2013 IEEE Aerospace Conference, 2013, pp. 1-13, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/document/6496853

[177] B. N. Tie and J. Bolluijt, "Measuring project complexity," *2014 9th International Conference on System of Systems Engineering* (SOSE), 2014, pp. 248-253, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/document/6892496

[178] G. Schuh, S. Rudolf and C. Mattern, "Conceptual framework for evaluation of complexity in new product development projects," 2016 IEEE International Conference on Industrial Technology (ICIT), 2016, pp. 1022-1027, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/document/7474894

[179] C. Fang, F. Marle and M. Xie, "Applying importance measures to risk analysis in engineering project using a risk network model," in *IEEE Systems Journal*, vol. 11, no. 3, pp. 1548-1556, Sept. 2017, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/document/7441987

[180] C. Ellinas, N. Allan and A. Johansson, "Toward project complexity evaluation: a structural perspective," in IEEE Systems Journal, vol. 12, no. 1, pp. 228-239, March 2018, New York, NY, USA: IEEE. [Online]. Available: https://ieeexplore-ieee-org.libproxy.nps.edu/document/7482752

[181] S. Tamaskar, K. Neema and D. Delaurentis, "Framework for measuring complexity of aerospace systems," *Research in Engineering Design*, vol. 25, issue 2, New York, NY, USA: Springer, February 22, 2014. [Online]. Available: https://www.deepdyve.com/lp/springer-journals/framework-for-measuring-complexity-of-aerospace-systems-AK7meeeqhf

[182] M. A. Horning, R. E. Smith, and S. Shidfar, "Mission engineering and prototype warfare: operationalizing technology faster to stay ahead of the threat," Proceedings of the 2018 Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), August 7–9, 2018, Novi. Michigan, USA. [Online]. Available: http://gvsets.ndia-mich.org/documents/SE/2018/Mission%20Engineering%20and%20Prototype%20Warfar%20-%20Operationalizing%20Technology%20Faster%20to%20Stay%20Ahead%20of%20the%20Threat.pdf

[183] J. D. Moreland, "Mission engineering integration and interoperability," Naval Sea Systems Command, NSWC Dahlgren Division, Washington Navy Yard, DC, USA. [Online]. Available: https://www.navsea.navy.mil/Home/Warfare-Centers/NSWC-Dahlgren/Dahlgren-Resources/Leading-Edge/I-I-Leading-Edge/Moreland/

[184] A. Alice, "Gaps in the body of knowledge of systems," 22nd Annual INCOSE International Symposium, Rome, Italy, 9–12 July 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2012.tb01449.x

[185] Project Management Institute, "A guide to the project management body of knowledge," Sixth Edition, Newtown Square, PA, USA: Project Management Institute, 2017. [Online]. Available: https://www.engineeringmanagement.info/2018/08/pmbok-guide-sixth-edition-summarized-pdf.html

[186] E. Bradner, G. Mark, and T. D. Hertel, "Effects of team size on participation, awareness, and technology choice in geographically distributed teams," *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS)*, p. 10, January 6–9, Big Island, Hawaii, USA, 2003. [Online]. Available: https://d2f99xq7vri1nk.cloudfront.net/legacy_app_files/pdf/2003%20Bradner%20HICSS.pdf

[187] T. J. Allen and G. Henn, "The organization and architecture of innovation: managing the flow of technology," p. 152, Burlington, MA, USA: Butterworth-Heinemann, 2006.

[188] K. Day and M. Hancock, "DIA hits significant construction cost overruns," CBS4 Denver, USA, April 29, 2014. [Online]. Available: https://denver.cbslocal.com/2014/04/29/dia-hits-significant-construction-cost-overruns/

[189] J. P. Byrne, "Project management: how much is enough?" PM Network, vol. 13, pp. 49–52, Newtown Square, PA, USA: PMI, February 1999. [Online]. Available: https://www.pmi.org/learning/library/project-management-much-enough-appropriate-5072

[190] T. Mochal, "Project management," Tech Decision Maker, TechRepublic Premium, August 5, 2008. [Online]. Available: https://www.techrepublic.com/blog/tech-decision-maker/use-this-process-to-estimate-a-projects-effort-hours/

[191] "What is CMMI?" Broadsword Solutions Corporation, San Diego, CA, USA. [Online]. Available: https://broadswordsolutions.com/what-is-cmmi/

[192] E. Ng'ang'a and I. Tonui, "A survey on software sizing for project estimation," *International Journal of Software Engineering and Knowledge Engineering*, vol. 5, Issue 4, Hackensack, NJ, USA: World Scientific Publishing, January 2015. [Online]. Available: https://www.researchgate.net/publication/275837351_A_Survey_on_Software_Sizing_for_Project_Estimation

[193] H. Van Heeringen, "Software size measures and their usefulness for software project estimation," ICEAA, June 8, San Diego, USA, 2015. [Online]. Available: http://www.iceaaonline.com/ready/wp-content/uploads/2015/06/SW14-Presentation-VanHeeringen-Software-Size-Measures.pdf

[194] M. Alfadel, A. Kobilica and J. Hassine, "Evaluation of Halstead and cyclomatic complexity metrics in measuring defect density," *2017 9th IEEE-GCC Conference and Exhibition (GCCCE),* pp. 1–9, May 8–11, Manama, Bahrain: IEEE, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8447959

[195] M. A. E. Latif, S. Kholeif, and K. A. Elwahab, "Identify and manage the software requirements volatility," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 5, West Yorkshire, UK: SAI, 2016. [Online]. Available: https://thesai.org/Downloads/Volume7No5/Paper_10-Identify_and_Manage_the_Software_Requirements_Volatility.pdf

[196] N. Nurmuliania, D. Zowghi and S. P. Williams, “Requirements volatility and its impact on change effort: evidence-based research in software development projects,” *Proceedings of the 2004 Australian Software Engineering Conference*, Sydney, Australia, 2004. [Online]. Available: https://pdfs.semanticscholar.org/55e3/af26a74cc268cd79ab67141894d5a27188f5.pdf

[197] G. B. Monk, “Integrated product team effectiveness in the department of defense,” Master thesis in Management, Naval Postgraduate School, Monterey, CA, USA, March 2002. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a402700.pdf

[198] Wikipedia. “Dependency injection.” [Online]. Available: https://en.wikipedia.org/wiki/Dependency_injection

[199] J. H. Panchal, C. J. J. Paredis, J. K. Allen, and F. Mistree, “Managing design process complexity: a value-of-information based approach for scale and decision decoupling,” paper no. DETC2007-35686, pp. 633–647, ASME Digital Collection, May 20, 2009. [Online]. Available: https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings-abstract/IDETC-CIE2007/48078/633/330304

[200] C. Haskins (CSEP and editor), *INCOSE Systems Engineering Handbook ver. 3.2*, INCOSE-TP-2003-002-03.2, San Diego, CA, USA: INCOSE, January 2010. [Online]. Available: https://www.incose.org/products-and-publications/se-handbook

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California