
# NAVAL POSTGRADUATE SCHOOL

**MONTEREY, CALIFORNIA**

# THESIS

**A PROCESS ARCHITECTURE MODEL THAT SUPPORTS COST AND EFFORT ANALYSIS FOR AGILE SOFTWARE DEVELOPMENT PROJECTS**

by

Robert M. Gallerani and Joseph M. Simonetti

September 2018

Thesis Advisor: Kristin M. Giammarco
Co-Advisor: Raymond J. Madachy

THIS PAGE INTENTIONALLY LEFT BLANK

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



| **1. AGENCY USE ONLY** *(Leave blank)* | **2. REPORT DATE** September 2018 | **3. REPORT TYPE AND DATES COVERED** Master's thesis |
|---|---|---|

**4. TITLE AND SUBTITLE**
A PROCESS ARCHITECTURE MODEL THAT SUPPORTS COST AND EFFORT ANALYSIS FOR AGILE SOFTWARE DEVELOPMENT PROJECTS

**5. FUNDING NUMBERS**

**6. AUTHOR(S)** Robert M. Gallerani and Joseph M. Simonetti

**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)**

The purpose of this thesis is to understand disparate organizational standard operating procedures (SOPs) covering agile software development and supporting functions, including business and technical feasibility analysis, contracts development, and personnel assessment. On the basis of SOP analysis, we developed a discrete-event software process simulation model of the architecture using Lifecycle Modeling Language (LML) action diagrams with the Model-Based Systems Engineering tool Innoslate. The action diagrams unify the SOPs to support both process architecture development and the ability to simulate actions independently or as a whole. The architecture illustrates that, in addition to the core function of software design and development, there are supporting functions that are necessary to successfully execute agile software development. The simulation model also serves as an accurate cost estimator for sprints. Historical data was available to calibrate model parameters for activity effort, staffing, and labor rates. The results of Monte Carlo simulations to forecast effort and cost for software sprints showed a high degree of accuracy against actuals. It is a viable alternative to other estimation methods and also provides risk assessment. The process model can be further calibrated and dynamically extended to support agile software development.

**14. SUBJECT TERMS**
model-based systems engineering, software process modeling and simulation, agile software development, software cost estimation, discrete-event simulation, process architecture, standard operating procedure, Innoslate

**15. NUMBER OF PAGES**
199

**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.**

# A PROCESS ARCHITECTURE MODEL THAT SUPPORTS COST AND EFFORT ANALYSIS FOR AGILE SOFTWARE DEVELOPMENT PROJECTS

Robert M. Gallerani
Civilian, Department of the Navy
BS, James Madison University, 2007

Joseph M. Simonetti
Civilian, Department of the Navy
BS, National University, 2002
MBA, University of Maryland University College, 2012

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT**

from the

**NAVAL POSTGRADUATE SCHOOL**
**September 2018**

Approved by: Kristin M. Giammarco
Advisor

Raymond J. Madachy
Co-Advisor

Ronald E. Giachetti
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

The purpose of this thesis is to understand disparate organizational standard operating procedures (SOPs) covering agile software development and supporting functions, including business and technical feasibility analysis, contracts development, and personnel assessment. On the basis of SOP analysis, we developed a discrete-event software process simulation model of the architecture using Lifecycle Modeling Language (LML) action diagrams with the Model-Based Systems Engineering tool Innoslate. The action diagrams unify the SOPs to support both process architecture development and the ability to simulate actions independently or as a whole. The architecture illustrates that, in addition to the core function of software design and development, there are supporting functions that are necessary to successfully execute agile software development. The simulation model also serves as an accurate cost estimator for sprints. Historical data was available to calibrate model parameters for activity effort, staffing, and labor rates. The results of Monte Carlo simulations to forecast effort and cost for software sprints showed a high degree of accuracy against actuals. It is a viable alternative to other estimation methods and also provides risk assessment. The process model can be further calibrated and dynamically extended to support agile software development.

THIS PAGE INTENTIONALLY LEFT BLANK

# TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF FIGURES

# LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| AMBIA | architecture model-based interoperability assessment |
| C2IS | Command, Control, and Intelligence Systems |
| C4ISR | command, control, communications, computers, intelligence, surveillance, and reconnaissance |
| CM | configuration management |
| COCOMO | Constructive Cost Model |
| CMU SEI | Carnegie Mellon University Software Engineering Institute |
| DR | deficiency report |
| ECR | engineering change request |
| ERB | engineering review board |
| GUI | graphical user interface |
| I/O | input / output |
| ICAM | integrated computer-aided manufacturing |
| ICOMS | inputs, controls, outputs, and mechanisms |
| IDEF0 | ICAM definition for function modeling / integrated definition for function modeling |
| IEEE | Institute of Electrical and Electronics Engineers |
| INCOSE | International Council on Systems Engineering |
| LML | Life cycle Modeling Language |
| MBSE | model-based systems engineering |
| MMRE | mean magnitude relative error |
| RE | relative error |
| SME | subject matter expert |
| SOP | standard operating procedure |
| SPAWAR | Space and Naval Warfare |
| SPEC | Systems and Proposal Engineering Company |
| SSC Pacific | SPAWAR Systems Center Pacific |

THIS PAGE INTENTIONALLY LEFT BLANK

# EXECUTIVE SUMMARY

In an effort to determine how to improve performance and management of agile software development projects within the context of a government organization, the Space and Naval Warfare (SPAWAR) Systems Center (SSC) Pacific Command, Control, and Intelligence Systems (C2IS) Division gathered data and heuristics over the past decade from its internal projects as well as academia, and industry. This led to the development of standard operating procedures (SOPs) for management and execution of agile software development projects. As stand-alone artifacts, these SOPs are not conducive for widespread acceptance and utilization due to their stove-piped and complex nature. To address this issue, we developed a holistic integrated model-based systems engineering (MBSE) process architecture to provide a consistent, integrated approach for performing agile software development in the organization. The process model architecture was generated using four key SOPs, which include aspects of business and technical feasibility, personnel, contract development, and agile software development.

Five key takeaways were obtained from the creation of the process architecture model. First, by following a Life cycle Modeling Language (LML) approach, the development of integrated definition for function modeling (IDEF0) diagrams from action diagrams follow a natural progression that ensure a thorough capture of the requisite SOP functions. Second, the LML action diagrams provide an integrated architecture with proper form and function mapping, which provides a means for addressing the issues with stove-piped SOPs. Third, the core SOPs adequately captured the processes for the business, contracting, and personnel to develop the LML representations of action diagrams and functional architecture. Fourth, successful application of metrics to simulate software development sprints within the architecture produced a model that accurately reflects the agile software development environment. Fifth, the discrete-event simulations provide insight into the possibilities of using MBSE approaches to support cost and schedule estimates for agile software development within the C2IS division. The outputs were validated based on a battery of model validation tests and statistical analysis against

historical data. The process architecture provides an extensible model that can be adapted for other software development projects.

For future work relative to this thesis, additional research can be performed to further assess incorporation of functions and form for deployment, maintenance, and retirement of software within the holistic architecture. With respect to quantifiable data for Monte Carlo simulations, additional metrics can be obtained for business, contracts, and personnel functions within the C2IS division to expand the scope of simulations. We discovered that metrics were only available for the number of developers involved in a software sprint, but not pre-sprint or post-sprint. Collecting additional metrics for current simulations can further increase the fidelity of the model. Additionally, while continuous integration and test are adequately covered, the SOPs could be further expanded to include additional activities such as sprint planning, execution, and review.

The high-level action and IDEF0 diagrams provide context for the major components of the architecture. Innoslate, which is owned by Systems and Proposal Engineering Company (SPEC) Innovations, is the MBSE tool used to create the integrated process architecture diagrams. The diagrams capture SOP details and integrate their guidance into a discrete-event model of the process architecture. The top-level action diagram is partitioned into four major actions (see Figure 1). The IDEF0 diagrams utilize the activities performed in action diagrams to graphically display the associated forms for the functional architecture (see Figure 2). The four main functions in the IDEF0 architecture align to the entities in the action diagram for business analysis, developing contracts, assessment of personnel, and agile software development. The IDEF0 diagram illustrates numerous inputs, outputs, controls, and mechanisms that comprise an architecture view, whereas the action diagrams capture the high-level process flow. The context of an IDEF0 “establishes the boundaries of the system or organization being modeled by defining the inputs and controls entering from external systems and the outputs being produced for external systems” (Buede 2009, 67).

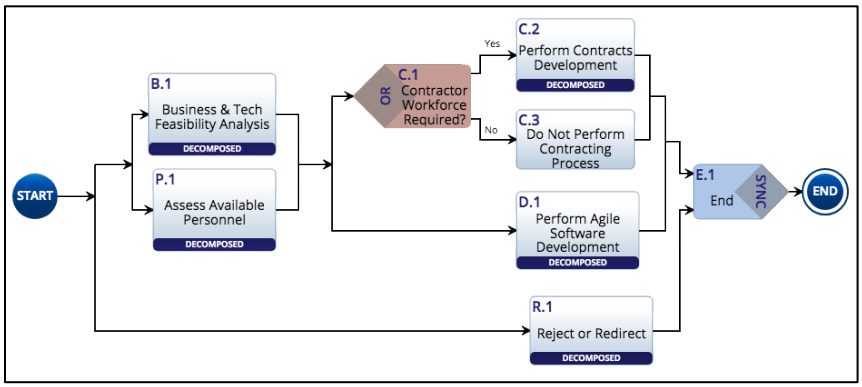



Figure 1. Top-Level Action Diagram

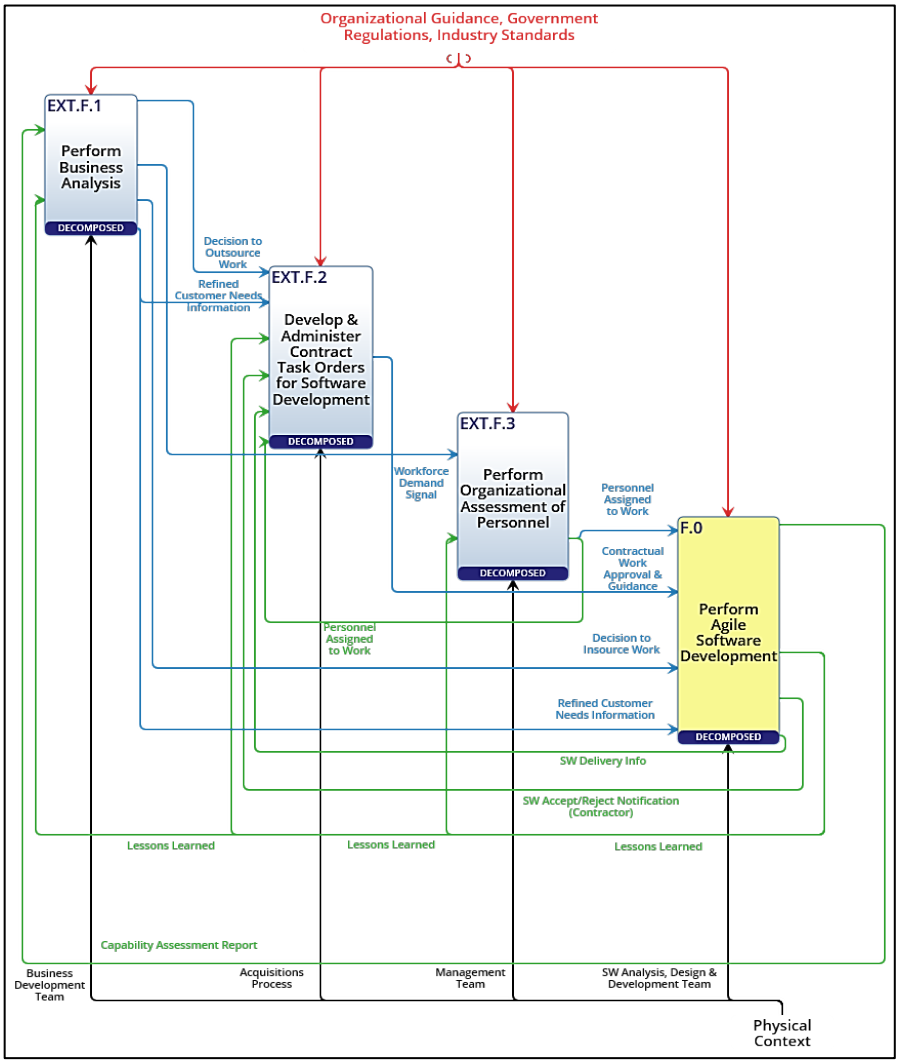



Figure 2. Architecture to Enable Agile Software Development IDEF0

Action diagrams for "conduct software sprint" (D.1.3) were created within "perform agile software development" (D.1) in Figure 1. The action diagrams for D.1.3 were used to perform effort and cost-based Monte Carlo simulations. The scope of the simulations is limited to data from a single software development project within the C2IS division. This historical data was used to calibrate activity effort, staffing, and labor rates. The inputs and outputs of the model for D.1.3 are illustrated in Figure 3.

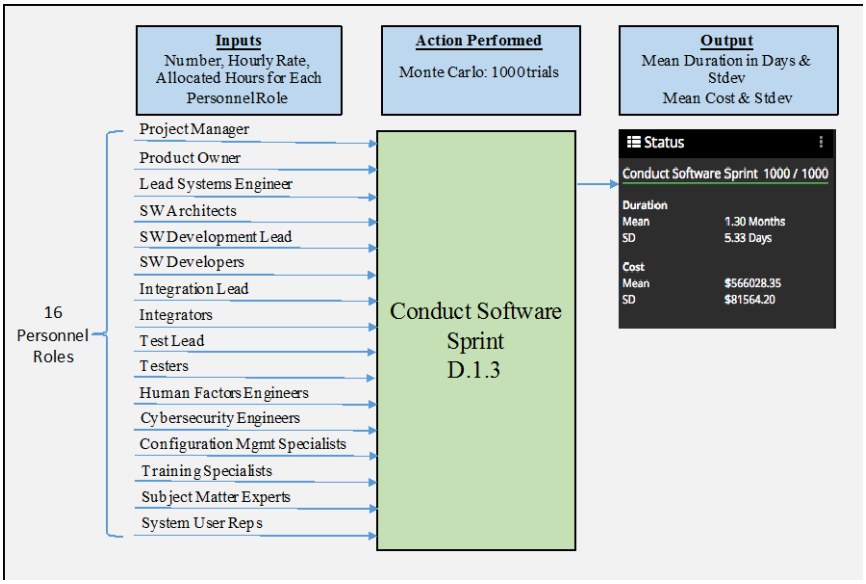



Figure 3. Model input parameters and outputs for Conduct Software Sprint (D.1.3)

Models were validated using 14 different structure and behavior tests covering "suitability for purpose, consistency with reality source, and utility and effectiveness of a suitable model" (Madachy 2008, 119–121). Measures including magnitude of relative error (MRE), mean magnitude relative error (MMRE), coefficient of determination ($R^2$), and PRED were used to analyze the cost prediction accuracy of the simulation. The MMRE provides the average of the MRE values for a dataset. For MRE values, a lower percentage value indicates a closer alignment between predicted and actual historic values. The "coefficient of determination shows how much variation in dependent variable is explained" (Rosa et al. 2017, 34). The PRED(20) and PRED(30) values represent the

percentage of estimates that have an MRE percentage value below 20% and 30%, respectively. For PRED values, a higher percentage indicates better performance.

Scatter plots were generated from the output of Develop Software Code (D.1.3.3.5.17) to compare the actual and simulated data points for cost in dollars and effort in person-hours (see Figure 4). The model prediction accuracies for "develop software code," per our model simulation produced an $R^2$ of 68.5%, an MMRE of 10.3%, and PRED(20) of 90%. For the corresponding cost simulation, our model yielded an $R^2$ of 59.3%, an MMRE of 12.8%, and PRED(20) of 85.7%. Traditional cost models generally attain a PRED(30) no better than 70%. This represents the upper limit of software cost models.

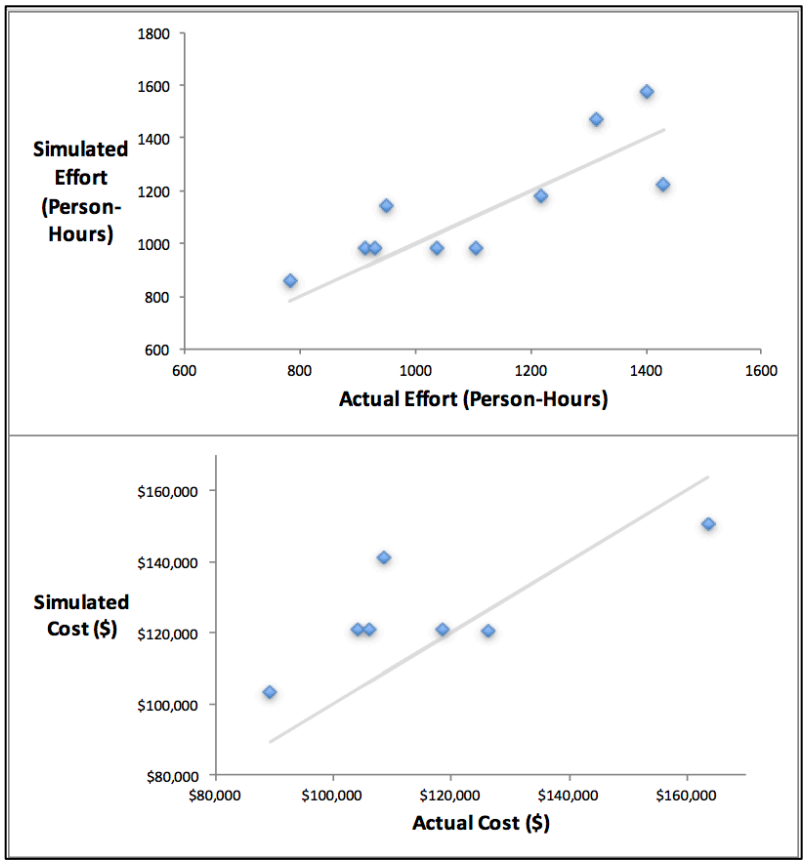



Figure 4. Actual and Simulated Output Comparison for Develop Software Code (D.1.3.3.5.17)

## References

Buede, Dennis M. 2009. *The Engineering Design of Systems: Models and Methods*. 2nd Ed. Hoboken, NJ: John Wiley & Sons.

Madachy, Raymond J. 2008. *Software Process Dynamics*. Piscataway, NJ: Wiley-IEEE.

Wilson, Rosa, Raymond Madachy, Bradford Clark, and Barry Boehm. 2017. "Early Phase Cost Models for Agile Software Processes in the U.S. DoD." Paper presented at *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement,* Toronto, ON, Canada, November, 2017. doi:10.1109/ESEM.2017.10.

# ACKNOWLEDGMENTS

Our team would like to thank the following persons for their contributions toward this thesis:

Dr. Kristin Giammarco, for her outstanding guidance and direction in development of the thesis and its architecture model.

Dr. Raymond Madachy, for his thesis guidance and attention to detail in developing the quantification of metrics and corresponding simulation.

The software developers and systems engineers within the SSC Pacific Command and Intelligence Systems Division, for their provision of historic software development metrics, and the Command leadership who supported, backed, and encouraged both students throughout this thesis and program.

Joseph Simonetti would like to thank and acknowledge his wife and daughter for their unconditional and amazing support and understanding throughout this process.

Robert Gallerani thanks his family and fiancée for their love and support throughout the thesis development process.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. INTRODUCTION

## A. BACKGROUND

Space and Naval Warfare (SPAWAR) Systems Center (SSC) Pacific, located in San Diego, CA, is part of the naval research and development establishment. SSC Pacific provides naval, joint, coalition partners with key capabilities in command, control, communications, computers, intelligence, surveillance, and reconnaissance (C4ISR), space, and cyber security. To improve performance and management of agile software development projects, the Command, Control, and Intelligence Systems (C2IS) Division within SSC Pacific gathered data and heuristics over the past decade to develop 36 standard operating procedures (SOPs). The SOPs are based on industry and academic best practices, including those derived from the Carnegie Mellon University Software Engineering Institute (CMU SEI), Institute of Electrical and Electronics Engineers (IEEE), agile software development processes, and experience from software development subject matter experts. They also implement continuous process and product improvement by incorporating lessons learned based on past project success and failure. This thesis captures SOP content in the context of a process architecture using a model-based systems engineering (MBSE) approach that encompasses a "formalized application of modeling to support…design, analysis, verification and validation activities" (INCOSE 2007). The SOPs include systems engineering considerations for agile software development. Systems engineering must incorporate software development as a consideration to achieve system-level results.

The 36 SOPs provide guidance for execution and management relative to agile software development. These SOPs include agile software development assets such as business and technical feasibility, personnel, contracts, configuration management (CM), cyber security, engineering change requests (ECRs), deficiency reports (DR), continuous integration and testing, and customer involvement. Within these 36 SOPs, preliminary analysis identified significant redundancy and out of scope instructions, which reduced the number of applicable procedures to 18 SOPs. The 18 SOPs selected had characteristics that indicated the organization could benefit near-term from adoption of 18 core SOPs that are

already generally applicable to software development projects within the Command, Control, and Intelligence Systems Division. Although these 18 SOPs showed signs of providing beneficial guidance, they were focused on specific projects and consequently yielded inextensible, project-specific guidance, which does not facilitate broader adoption. Consequently, these 18 SOPs were further distilled to four primary SOPs that have broad organizational applicability. These final four SOPs focus on business and technical feasibility, personnel, contract development, and agile software development. These four SOPs provide the primary source for assessing and generating the top-level architecture content.

The process architecture incorporates the interconnection of SOP guidance, dependencies, and their boundary constraints. Innoslate, which is owned by Systems and Proposal Engineering Company (SPEC) Innovations, is the MBSE tool utilized for creating the integrated process architecture. We used MBSE to enable visualization of data and processes through models, yields traceable links within the process architecture, and provides "a better way of creating, managing, and verifying engineering data than textual specifications" (Douglass 2016, 23). Life cycle Modeling Language (LML) representations are utilized to capture the corresponding details of the SOPs. Examples of LML representations are action diagrams and integrated computer-aided manufacturing (ICAM) definition for function modeling / integrated definition for function modeling (IDEF0) diagrams. An additional feature of Innoslate is the ability to conduct effort-based simulations using action diagrams. Action diagrams within Innoslate can be updated with real-world metrics, which enables to simulation of specified actions to produce activity cost and duration estimates. In the case of software sprint effort, quantifiable data gathered from the C2IS division was applied to applicable action diagram for use during simulations. Within the context of the proposed architecture, the scope of a given simulation is limited to a single software development project.

"Process modeling is representing a process architecture, design, or definition in the abstract" (Madachy 2008, 22). By modeling the SOPs for the C2IS division, understanding of the associated interrelated processes becomes adeptly understood by visualizing the process architecture in a model. "The power of models increases

dramatically as they become more explicit and commonly understood by people; hence, process modeling is ideally suited for organizational improvement" (Madachy 2008, 29). This benefit of modeling is compounded with the ability to perform model-based simulations. "Simulation is an efficient communication tool to show how a process works while stimulating creative thinking about how it can be improved" (Madachy 2008, 24). Processes need to be accurately modeled within the MBSE process architecture so that simulations reflect processes as they would occur in practice. Accurate models yield realistic simulations, which provide output that has greater fidelity. Since the simulations for this process model architecture vary stochastically for each run based on a multitude of manually or statistically selected decision paths, "the result variables are best analyzed statistically (e.g., mean, standard deviation, distribution type) across a batch of simulation runs. This is termed Monte Carlo simulation or Monte Carlo analysis" (Madachy and Houston 2018, 4). Understanding how the architecture should reflect process modeling and simulation helps in designing the architecture so the SOPs are effectively modeled in a logical flow. By designing the process architecture so that the SOPs can be run through model-based simulations, the flow of process inputs and outputs between actions is more acutely understood and sequenced accordingly in the process architecture. Understanding the dynamics with software process modeling and simulation provides context to the overall architecture.

## B. PROBLEM STATEMENT

The origin of our proposed SOP-based process architecture stems from C2IS division experience with past challenges regarding software development projects. "System architecture is the embodiment of concept, the allocation of physical/ informational function to the elements of form, and the definition of relationships among the elements and with the surrounding context. If we are to deliver value with the system we build, it must have good architecture" (Crawley, Cameron, and Selva 2017, 110). Additional challenges that can affect software development efforts are inherited from issues with contractual language, stove-piped design and development processes, and inconsistent application of lessons learned for the workforce. These deficiencies increase the likelihood of incurring issues associated with corresponding schedule, cost, and

technical performance risks. To help mitigate these risks, four SOPs are available to help guide SSC Pacific software development projects. The inherent issue of trying to utilize the four SOPs as stand-alone artifacts is that they have a stove-piped and complex nature that is nonconductive for widespread acceptance and utilization. To investigate remediating this issue, we performed an iterative analysis on the SOPs to develop a holistic integrated architecture model, which integrates corresponding inputs, outputs, controls, and mechanisms. A holistic integrated architecture considers the relationships within and between other systems (Crawley, Cameron, and Selva 2017, 20). The development and successful simulation of supporting action diagrams preceded the validation of the integrated architecture. To ensure the process architecture model performs as intended to match the intent of the SOPs, model testing and application of verification and validation tests are required determine if the model can accurately predict cost and effort, assess the validity of the model.

## C. PROJECT GOALS AND DELIVERABLES

One of the goals of the C2IS division is to provide elegantly engineered command and control capabilities for naval, joint, and national level customers. To achieve this goal, it is paramount that a well thought out, integrated architecture that accounts for business and technical feasibility analysis, contracting processes and development, software design and development processes, and allocation of personnel to perform the work. Since the current SOPs are stand-alone and do not readily present their interdependencies with other SOPs, there is an opportunity to develop an architecture that demonstrates critical interdependencies. The action and IDEF0 architecture diagrams generated for this project will provide the C2IS division with a holistic architecture that yields an integrated view of the current stove-piped SOPs.

The MBSE process architecture we developed is intended to provide an easily understood flow of interconnected activities defined within the SOPs. In this capacity, the architecture could assist a new project manager in facilitating the execution of an agile software development cycle within the C2IS division. Additionally, one can feed the process architecture model in Innoslate the statistical cost and duration values collected

from software development activities in the C2IS division to perform corresponding simulations. The process model can perform discrete and Monte Carlo simulations using this data to generate output for estimating cost and duration of software development processes. These outputs are useful to assist software developers and managers in assessing the accuracy of his or her effort and cost estimates. An additional effort encompasses validating and verifying the model through use of behavior and structure tests. Further collection of actual software development costs and durations from the C2IS division will help improve the model's ability to simulate processes accurately. Improved simulation output has potential to bypass the need to expend actual resources in learning by trial and error. Given the prospective benefits of this architecture and simulation-based action diagrams, the results of this project will be used for potential adoption by other software centric projects within the C2IS division.

## D. ASSUMPTIONS AND CONSTRAINTS

To effectively convey understanding of a system or process, "it is important to define the system under consideration by specifying its limits, boundaries, or scope" (Blanchard, Wolter, and Fabrycky 2011, 5). The architectural design process entails taking system boundaries into consideration and clarifying what is inside and outside those boundaries (Crawley, Cameron, and Selva 2017, 24). Modeling techniques such as IDEF0 and flow diagrams can help better define system boundaries (Buede 2009, 144). The system boundaries for the architecture within this thesis provide a definitive scope for the high-level functions and forms. The architecture for this thesis was generated using action and IDEF0 diagrams in the MBSE tool, Innoslate. To provide boundaries with respect to the scope of the architecture, this thesis is limited to content within the four primary SOPs. The first SOP addresses major capability assessment gating functions. The second SOP addresses continuous software integration. The third SOP handles deficiency reporting, and the last SOP includes the control board process. Guidance from these SOPs is integrated into the top-level action diagram and IDEF0, which focus on the following four major actions: business and technical feasibility analysis; assess available personnel; perform contracts development; and perform agile software development.

Within the assets and actions aligned to the top-level architecture for business and technical feasibility analysis, there are several constraints to the primary functions performed within the C2IS division. The primary functions that are assumed within the business analysis include those dealing with customer demand, needs analysis, project planning, and work acceptance. Most business and technical feasibility functions are captured within the system boundary. However, there are myriad external systems and policies that constrain how business and technical functions are perform, which include organizational guidance, government regulations, and industry standards. For example, the work acceptance process will ensure work to be performed aligns to the organization's mission, and work that does not align, will not be permitted to proceed.

The process for assessing available personnel includes actions to perform an initial assessment of existing personnel and processes for performing a personnel selection from existing personnel. The boundary does not include hiring of external or new employees and the human resources process that accompanies onboarding new employees. The boundary also does not extend to educational or other research institutions. The main boundary assumption is that existing employees will be assigned or redirected to new work and supported by a contractor workforce. New employees can be hired; however, that process is outside the scope of this thesis.

Contractual development is contained to the task order level as opposed to the basic contract level. The C2IS division uses multiple award contracts (MACs) versus single-award contracts. Within a MAC environment, task orders are awarded against each basic contract. The contractual process modeled and incorporated into the architecture is limited to task orders with the assumption that the basic contract has already been awarded. Modeling the federal acquisition regulatory process and statutes that govern a single award or MAC contract is outside the boundary of this thesis. In addition, within the contractual development process, the boundary does not extend to include the detailed processes of contracting specialists or legal representatives that compete and award the contracts. Those personnel interface with other external systems that are beyond the scope of this thesis.

The primary architecture function of performing agile software development includes the sub-functions of performing capability assessments, software design, software

development and review, configuration management services, software quality engineering, and continuous integration and test. Within this primary function, design and development tools and infrastructure used for the architecture can be contained within the system boundary; however, they may cross the architecture boundary into an external system. For example, if one selected a server-based development implementation, the tools used within the server would be considered within the boundary of the system, but the server provider and their supporting systems would be considered outside the boundary of the architecture. Assets reflecting local instantiations of agile software development tools and infrastructure would be considered part of the architecture boundary. In essence, the physical boundary of the system may change with the implementation method chosen. "Boundary conditions mediate the flow of energy, matter, material wealth, and information (EMMI) across interfaces at boundaries" (Langford 2012, 42). On-premise hardware may be utilized, and a cloud-based infrastructure may also be utilized. From an architecture perspective, the functions performed are more critical than the physical asset performing them. There are additional activities that occur after software is developed and a decision to release the software is made. Examples such as formal operational testing, deployment of the software via download or manual delivery, and post deployment support are outside the boundaries of and not within scope of the models developed for this thesis.

Another factor to consider within the parameters of the architecture is that the performing assets are not prescriptive with respect to the personnel performing the action. An example of the non-prescriptive nature of the architecture assets is that a software designer can also be a software quality engineering team member in a separate activity. The software designer and software quality engineering team member in this example is not mutually exclusive. This holds true in the architecture for many additional activities. An additional constraint implemented in this architecture is derived from the heuristic of only specifying one mechanism per function. This adheres to the best practice described in Giammarco's 2012 architecture model-based interoperability assessment (AMBIA) dissertation, which states, "every exchanged resource between any two performers is subject to some rule that constrains those performers" (17). In the case of the subject

functional architecture, there are overarching controls of organizational guidance, government regulations, and industry standards.

Quantifiable metrics were gathered for specific process parameters of the Innoslate action diagrams to enable realistic simulation within the MBSE Innoslate tool. Due to limitations in gathering and applying historical metrics from the C2IS division, real-world metrics are only applied to the software sprint section of the agile software development Innoslate architecture. Metrics were available for the planned and actual number of developers involved in a software development sprint. We discovered that projects were not tracking the planning work leading up the sprint and the review work after the sprint with as much details as the actual sprint. The model developed establishes the framework to enable collection of metrics within the model for future use.

## E. OVERVIEW OF CHAPTERS

After he background, purpose of the SOPs, and the purpose of the corresponding architecture created for this thesis are reviewed in Chapter I, Chapter II delves into software process models, applicability to existing research, and the methodology utilized to assess the standard operating procedures and generate the corresponding architecture. The chapter describes how the MBSE tool Innoslate was utilized to construct the architecture and explores the modeling framework utilized, which focuses on the agile software development process.

The architecture and action diagrams focus on four holistic functions that enable agile software development within a government environment: business and technical feasibility analysis; assess available personnel; perform contracts development; and perform agile software development. These details are captured in Chapter III, which focuses on the action and IDEF0 diagrams. Within Chapter III, the architecture is decomposed from the top-level action diagram and IDEF0 model down to their respective component parts. This chapter describes the actions and assets within each diagram and it explains their corresponding inputs, outputs, controls and mechanisms. In aggregate, Chapter III provides a holistic agile software development architecture.

Chapter IV focuses on model usage and the details behind simulating the action diagram architecture and quantifying corresponding actions. It describes the architecture and corresponding modifications made to simulate the diagrams in Innoslate. Quantification of metrics for simulation focuses on the Software Sprint process within the agile software development cycle and ties the metrics to real-world data obtained from a software development project in the C2IS division within SSC Pacific.

Chapter V provides analysis regarding model prediction measures and threats to model validity. Discussion includes an analysis of model testing and application of verification and validation tests to determine if the model can accurately predict effort and cost assess the validity of the model. The model is discussed in the context of traditional post-calibrated models and applicability of use in early phase cost estimating.

The conclusion summarizes the findings with respect to an architecture model for agile software development within the organization. After revisiting the overall findings based on the thesis statement, recommendations based on the findings are offered. Following delivery of recommendations, the conclusion provides a summary of topics that were out of scope for this project, which could be researched as future work.

THIS PAGE INTENTIONALLY LEFT BLANK

# II. RELATED WORK AND APPROACH

## A. INTRODUCTION

This chapter discusses software process models and software process modeling. To put the modeling into context, we discuss applicability to related research in the area of software process modeling and simulation. In addition, the modeling approach used to develop action diagrams and IDEF0 diagrams is discussed. Section B discusses software process models and process modeling. Rationale is provided as to why the agile framework was selected over other software process models. Section C discusses software process modeling. For this research, we implemented a discrete event software process modeling approach. Section D provides a literature review of related efforts applicable to this thesis. Section E outlines the methodologies utilized to create the action and IDEF0 diagrams described in Chapter III and model usage described in Chapter IV in order to convey the content of the organizational SOPs for agile software development.

## B. SOFTWARE PROCESS MODELS

Developing an effective architecture requires understanding the process model implemented within that process architecture. As such, one of the first steps taken is to select a process model that captures how the C2IS division creates and delivers software. Both prescriptive and agile software development process models were assessed. Since all C2IS division SOPs focused on use of an agile framework, this is the framework that we chose to implement. However, to put the agile process model architecture in context, it is beneficial to understand several other prescriptive process models in addition to the agile framework. Discussion for prescriptive process models provides background and context of numerous software process models. The agile process model is discussed within the context of how it is utilized within the organization.

### 1. Prescriptive Process Models

A prescriptive process model provides an organized means for developing new software by prescribing a specific structured process (Pressman 2010, 38). Examples of

prescriptive process models include the waterfall process model, the incremental process model, the spiral development process model, and the V-model (Pressman 2010, 39–46). Each of these models has its respective benefits and deficiencies. Understanding these process model types provides context relative to the agile process model used.

The waterfall process model provides "recognition of the feedback loops between stages, and a guideline to confine the feedback loops to successive stages to minimize the expensive rework involved in feedback across many stages" (Boehm 1988, 63). However, there are challenges with this process model. "A primary source of difficulty with the waterfall model has been its emphasis on fully elaborated documents as completion criteria for early requirements and design phases" (Boehm 1988, 63). The waterfall process model requirement for extensive documentation results in work that is not necessarily required and can consequently add avoidable effort and cost to software development projects. The waterfall model can be observed in Figure 1.

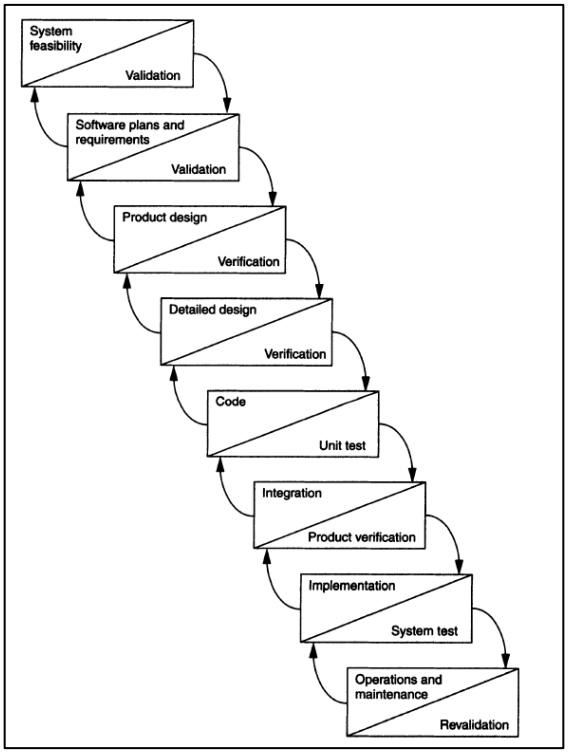



Figure 1. Waterfall Software Process Model. Source: Boehm (1988).

Another prescriptive process model is the incremental process model. Incremental process modeling takes “a relatively narrow slice of functionality through all activities to produce a version of the system that is then verified and validated, before incrementally adding the next slice of functionality” (Douglass 2016, 22). The incremental process model can be visualized as a cycle in which validated work is reintroduced to subsequent work as an iterative building process (see Figure 2). “The success of incremental and agile methods in software development is due largely to the ease with which software can be refactored” (Douglass 2016, 22). The incremental model and agile process model share in elements of success due to their ability to produce software quickly and to process rework. Developing and delivering software using an iterative approach also can help provide early insight into risk of defective software code. However, an assumption of the incremental approach is that it depends on up-front and available well-defined requirements and is referred to as “a ‘depth-first’ approach” (Douglass 2016, 22). This can lead to challenges from a systems architecture approach if all requirements are not available at the onset.

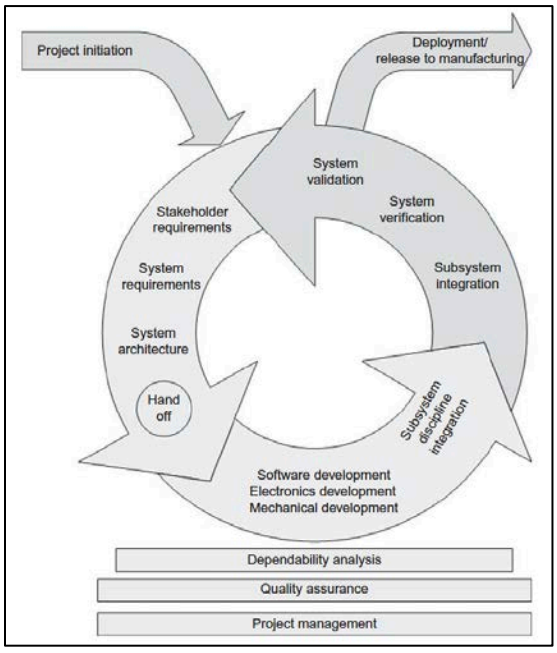



Figure 2. Incremental Life Cycle Process Model. Source: Douglass (2016).

The spiral process model is a prescriptive model that "can accommodate most previous models as special cases and further provides guidance as to which combination of previous models best fits a given software situation" (Boehm 1988, 64–65). In the spiral process model "the model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each portion of the product and for each of its levels of elaboration" (Boehm 1988, 65) (see Figure 3). Although the spiral process model is highly adaptable, it also has challenges. Its "three primary challenges involve matching to contract software, relying on risk-assessment expertise, and the need for further elaboration of spiral model steps" (Boehm 1988, 69–70).

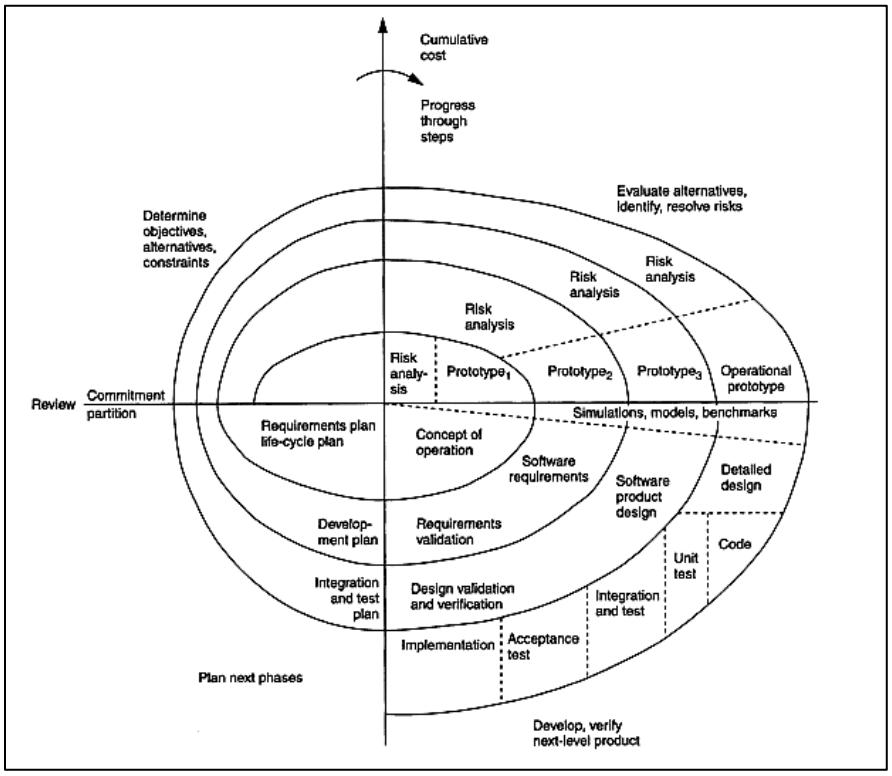



Figure 3. Spiral Software Process Model. Source: Boehm (1988).

Another prescriptive process model is the V-Model. "The V-model is an extended (or perhaps "bent") waterfall life cycle in which the activities on the left side of the "V" stipulate two different but related work products: a specification and its means of verification" (Douglass 2016, 20). The V-model process model can be visualized and

thought of as top-down system realization and bottom-up verification and validation (see Figure 4). “The V-model is a “breadth-first” approach in that each work product is assumed to be created in one activity and is expected to be complete, accurate, and correct at that point and forever more” (Douglass 2016, 20–21). This can be an issue for software projects with loosely or undefined requirements. In addition, Douglass points out that “not only are there unknowns when planning, some of the things you do know will change later. The V-model life cycle is notoriously resistant to changing customer needs, requirements, technologies, and staffing” (2016, 21).

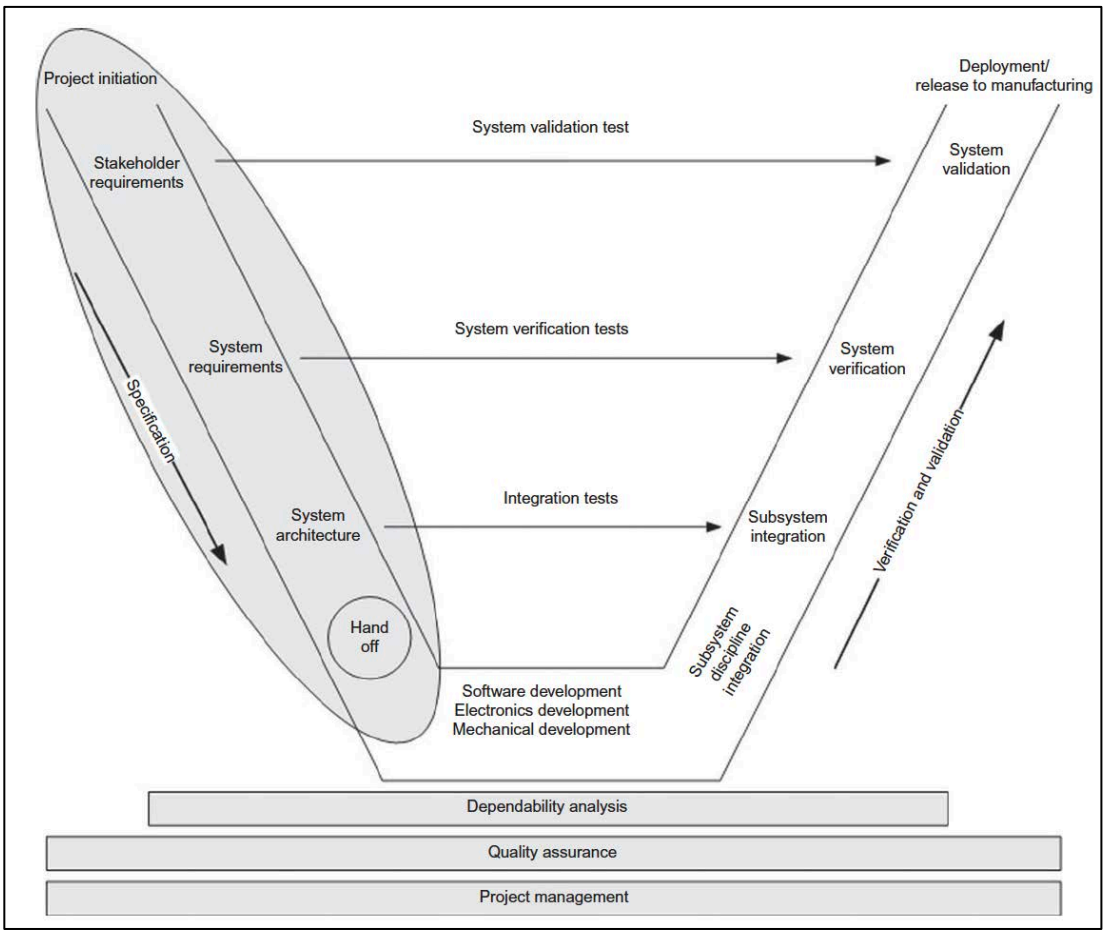



Figure 4. V-Model. Source: Douglass (2016).

## 2. Agile Framework

In agile software development, incremental development and delivery divides work "into meaningful slices of the total end result, delivered in gradually more complete versions" (Hayes and Miller 2017, 21). The agile framework can be further understood by examining its supporting tenants and characteristics. According to Madachy (2008, 37), there are four value propositions associated with the agile manifesto, which include "individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, [and] responding to change over following a plan" (Madachy 2008, 37). To clarify the agile framework approach to problem solving, the emphasis is placed "on 'solving the problem' by getting continuous customer and quality feedback rather than following the plan" (Douglass 2016, 44). The Agile Alliance specifies the following core principles that provide the impetus for agile software development and support the agile manifesto:

> Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. Business people and developers must work together daily throughout the project. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. Working software is the primary measure of progress. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Continuous attention to technical excellence and good design enhances agility. Simplicity—the art of maximizing the amount of work not done—is essential. The best architectures, requirements, and designs emerge from self-organizing teams. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. (Agile Alliance 2018)

The International Council on Systems Engineering (INCOSE) specifies that the key objective of agile development is "flexibility, and allowing selected events to be taken out of sequence when the risk is acceptable" (INCOSE 2011, 40). The aggregate results of these principles are emphasized by the Systems and Software Consortium (2007), which

upholds that successful implementation of the agile software development process ensures an earlier return on investment, an expedient delivery of working software, faster responses to customer changes, mitigation of schedule risk due to shorter delivery cycle, and higher productivity and quality. This agile framework is captured within action diagrams, which are discussed in detail in Chapters III and IV.

## C. SOFTWARE PROCESS MODELING

To execute effective simulations with meaningful output based on the process architecture captured in the process models, it is critical to understand software process modeling approaches and their suitability for simulating given scenarios. In the context of process modeling used, "simulation is a statistical sampling experiment in which models convert stochastic inputs into statistical data output" (Madachy and Houston 2018, 98). Process modeling simulations bring added value and insight to a process architecture, since "simulation is used to represent the behavior of systems, processes, or scenarios" (Madachy and Houston 2018, 3).

### 1. Overview of Process Modeling Approaches

There are several process modeling approaches to consider when deciding how to simulate a process. Process modeling approaches considered include "continuous modeling, discrete event simulation, and agent-based simulation" (Madachy and Houston 2018, 21) as well as hybrid process modeling. It is essential to understand the different types of process modeling techniques to ensure they are applied effectively for process architecture model simulation. Therefore, before proceeding with adopting a process modeling approach, it is important to weigh the assumptions, advantages, disadvantages, and tradeoffs of the different process modeling approaches. It is also important to keep these modeling techniques in context with respect to the C2IS division process architecture described in Chapter III. Discussion includes the concept of continuous, discrete-event, agent-based, and hybrid software process modeling.

### *a. Continuous Process Modeling*

In continuous process modeling "the continuous view does not track individual events; rather, tasks are treated 'in the aggregate' and systems can be described through differential equations" (Madachy 2008, 55). Additionally, "a continuous model shows how values of the attributes change as functions of time, computed at equidistant small time steps. These changes are typically represented by smooth continuous curves" (Madachy and Houston 2018, 3). Understanding the functionality of continuous process modeling provides further context relative to other process modeling approaches considered for this thesis. Within continuous process modeling, "system dynamics is the most widely used form of continuous simulation" (Madachy and Houston 2018, 58). System dynamics provides a modeling environment that can "facilitate human understanding and communication of the process, but does not explicitly consider automated process guidance or automated execution support" (Madachy and Houston 2018, 25).

### *b. Discrete-Event Process Modeling*

Discrete-event process modeling "approaches model each and every event. Their focus is usually on the flow of discrete entities without having feedback connections and the resulting internal dynamics" (Madachy 2008, 56). In the process architecture for this thesis, each simulated entity is depicted as a block within an action diagram. When the process model architecture is simulated, "the entities move through a system represented as a network of nodes, perform activities by using resources, and create events that change the state of a system" (Madachy and Houston 2018, 43). Discrete-event simulations result in variations that "occur instantaneously as the simulated time lapses and are reflected in the output as discontinuous fluctuations" (Madachy and Houston 2018, 3).

### *c. Agent-Based Process Modeling*

An additional approach to process modeling is agent-based modeling. This type of modeling represents "interacting entities, or agents. Agents have their own characteristics and can initiate actions, communicate with one another, and react to one another. Like discrete event simulation, agent-based models generally treat time as a series of discrete events" (Madachy and Houston 2018, 22). Agent-based processing modeling is an effective

"tool to study complex systems with many interacting entities and non-linear interactions among them. Emergent behaviors can result, which are patterns generated by the interactions of the agents, which are often unexpected" (Madachy and Houston 2018, 52). In this context of a process architecture, it is critical to know what comprises the model as this contributes the success of agent-based modeling.

#### *d. Hybrid Process Modeling*

An understanding of continuous and discrete-event process modeling is helpful to understand the dynamics of a hybrid process modeling approach. Hybrid process modeling is instantiated when its simulations "support both continuous and discrete event timing in the same model" (Madachy and Houston 2018, 22). Hybrid process modeling has an increased likelihood of being used when "no single modeling approach is well suited to all aspects of that situation" (Kellner, Madachy, and Raffo 1999, 15). By combining continuous and discrete processes, the hybrid process model can help perform a successful simulation where either of the processes alone would not be sufficient.

### 2. Process Modeling Approaches Advantages, Disadvantages, and Tradeoffs

#### *a. Advantages of Process Modeling Approaches*

There are advantages among each of the process modeling approaches discussed. One of benefits of continuous process modeling is that "continuous simulation models may also be applied to systems that are discrete in real life but where reasonably accurate solutions can be obtained by averaging values of the model variables" (Madachy and Houston 2018, 36). Additionally, continuous process modeling "accurately captures the effects of feedback" (Kellner, Madachy, and Raffo 1999, 15), and it provides a "clear representation of the relationships between dynamic variables" (Kellner, Madachy, and Raffo 1999, 15). For application of continuous models in practice, the "continuous models are useful in such areas as engineering design where well-established mathematical relationships give rise to models consisting of differential or algebraic equations" (Madachy and Houston 2018, 3).

There are several advantages associated with applying discrete-event process modeling. “Discrete event simulation views systems and processes as interconnected event-based flows of entities through queues and activities. This view corresponds well with intrinsic, measurable, real-world phenomena” (Madachy and Houston 2018, 45). Additionally, discrete-event simulation is “CPU efficient since time advances at events” (Kellner, Madachy, and Raffo 1999, 15). It also provides simulation flexibility since its “attributes allow entities to vary” (Kellner, Madachy, and Raffo 1999, 15). Discrete-event model processing is also advantageous since it “queues and interdependence capture resource constraints” (Kellner, Madachy, and Raffo 1999, 15). Additionally, “Discrete-event modeling has some advantages for product analysis because different attributes can be attached to individual entities like defects” (Madachy 2008, 273).

Advantages of agent-based process modeling are derived from its ability to utilize agents in environments. More specifically, “agent-based models have been primarily developed for socio-economic systems simulating the interactions of autonomous agents (both individual or collective entities such as organizations or groups) to assess their effects on the system as a whole” (Madachy and Houston 2018, 53). Using this holistic approach enables agent-based process modeling to simulate models in their respective environment more effectively.

Hybrid process modeling is beneficial for more complex models. This is particularly true in cases where “simulation software will no longer be so cleanly divided among continuous, discrete event, or agent-based methods. Applications will incorporate hybrid modeling to capture different perspectives and allow multiple-view insights” (Madachy 2008, 497). Additionally, “a hybrid approach combining continuous and discrete-event modeling is very attractive for product applications. It can model the creation of artifacts with attributes, modify those attributes based on system variables, and allow system variables to vary continuously” (Madachy 2008, 273). By combining the benefits of multiple process modeling approaches, hybrid process modeling can simplify more complex modeling cases by ingesting and simulating model variables with an appropriate modeling process.

### *b. Disadvantages of Process Modeling Approaches*

There are disadvantages that can be attributed to process modeling approaches as well. A shortfall of continuous process modeling is that "sequential activities are more difficult to represent" (Kellner, Madachy, and Raffo 1999, 15). Additionally, continuous process modeling has "no ability to represent entities or attributes" (Kellner, Madachy, and Raffo 1999, 15).

A primary disadvantage of discrete-event process modeling is that "continuously changing variables not modeled accurately" (Kellner, Madachy, and Raffo 1999, 15). Also, discrete-event process modeling has "no mechanism for states" (Kellner, Madachy, and Raffo 1999, 15). Without a mechanism for overall model states, it is possible that influential variables go unaccounted for in discrete-event simulations.

One of the main challenges associated with agent-based process modeling is that "agent-based modeling is relatively new without an extensive history of engineering usage like discrete event and continuous modeling" (Madachy and Houston 2018, 52–53). Without a history of metrics and prior work to reference, one can infer that agent-based modeling becomes harder to apply appropriately based on similar cases.

Although hybrid process modeling can draw from the benefits of other process modeling approaches, it also has a shortfall when it comes to assessing complexity. Since hybrid process modeling typically deals with more complex cases, "these situations may require understanding of complex feedback processes involving such interacting phenomena as schedule pressure, communication overhead, or numerous others" (Madachy 2008, 25). One can infer that increases in complexity in executing a simulation may result in a higher risk of error.

### *c. Tradeoffs of Process Modeling Approaches*

Based evaluation of the process modeling approaches for continuous, discrete-event, agent-based, and hybrid simulations, tradeoffs for the specified process modeling approaches are evident. Tradeoffs become apparent early in modeling a process architecture for the sake of data preservation, as exemplified by Madachy's (2008, 56)

work, which raises the question, “Are there discrete aspects that need to be preserved for the purpose of the study? If so, then a discrete or hybrid modeling approach may be better suited than system dynamics” (Madachy 2008, 56).

Additionally, understanding tradeoffs is important to identify which process modeling approach is right for a given model during its construction. The system model must account for process flow, interactions and interdependencies of model entities and resources, and characteristics and rules model behaviors. (Madachy and Houston 2018, 33). In some cases, tradeoffs become more evident based on validity of certain process models in context. “There are many systems that should be modeled as discrete because no continuous approximations are valid” (Madachy and Houston 2018, 3). There is also an opportunity to evaluate tradeoffs with respect to how performance of process activities is portrayed in the modeling. The level of model abstraction can influence whether a continuous model or discrete-event model is needed. Continuous models are usually more abstract than discrete event models (Madachy and Houston 2018, 33). When a more global view is required, a continuous model can be more appropriate. In contrast, when a more granular view is required that demonstrates step-by-step changes in model entities, a discrete-event model can be more appropriate (Madachy and Houston 2018, 33). If the behavior of individual objects in the context of how they interact and influence the collective model is required, then agent-based modeling is more appropriate (Madachy and Houston 2018, 33). Understanding these tradeoffs enables an improved capability in selecting the right process modeling approach for the right model simulation.

## 3. Selected Process Modeling Approach

A discrete-event process model is the most appropriate method for simulating the process model architecture for this thesis. Decision factors that influenced the decision to use discrete-event modeling include analysis of organizational SOPs for agile software development, and the advantages, disadvantages, and tradeoffs of the various process modeling approaches. “One of the first decisions to be made in modeling is choosing the modeling paradigm. This may be dictated by the modeler’s expertise in a particular paradigm or the paradigms supported by a simulation software package used in the

modeler's organization" (Madachy and Houston 2018, 21). The granularity of the models correlates to the level of detail within the SOPs, thus discrete-event modeling was better suited for the problem space. Our thesis uses the discrete-event process modeling to execute action diagrams in Innoslate based on the tools ability to simulate the corresponding process architecture in Innoslate using discrete and Monte Carlo simulations. This was the primary driver for using discrete process modeling to execute the Innoslate simulation for the action diagrams, discussed in detail in Chapters III and IV.

Our approach was to adopt a build-test-fix approach in a small incremental manner. This approach is instrumental in providing early insight into issues with the model as it is developed. Discrete-event process modeling also provides the ability to create starting points, stopping points, a logical and interconnected flow, with interdependencies between action entities (Madachy and Houston 2018, 45). When executing simulations in the process architecture, the concept of discrete-event process modeling was applied such that "attribute values are used to set activity durations and to route entities through a model. Variable values are updated with each event" (Madachy and Houston 2018, 21). A major deciding factor for using discrete-event process modeling was the fact that the process architecture tool, Innoslate, only supports discrete-event modeling at this time.

The problem domain presented by the C2IS SOPs and their corresponding action diagram process architecture in Innoslate yielded a prime opportunity to apply discrete-event process modeling methodology due to the nature of individual action diagram entities. Each action diagram entity in Innoslate provides the ability to update and adjust metrics, which can perform discrete or Monte Carlo simulations. The process architecture created in Innoslate maps directly to the detailed SOPs, which supports the simulations. The holistic discrete-event process modeling of this process architecture is key to the simulation success. "The quality of a simulation model depends on the structure of the model and on the quality of the model inputs. Every model input should be considered for its uncertainty" (Madachy and Houston 2018, 94). A more detailed level of tracking is enabled by using discrete-event process modeling for actions based on staff members.

Another modeling tool considered to develop the process architecture was Monterey Phoenix. "Monterey Phoenix (MP) is a framework for software system architecture and business process (workflow) specification based on behavior models" (Auguston 2018, 5). It provides insight into and helps answer questions regarding a "system's behavior, including such aspects as structure of behavior, dependencies between actions involved in the behavior, constraints on behaviors" (Auguston 2018, 6). Monterey Phoenix provides the ability to visualize those behaviors based on events that evolve over time, including subsystem behavior and their interaction (Auguston 2009, 3). Event attributes can include probability, duration, and cost. Monterey Phoenix "produces exhaustive set of all valid behaviors for a given scope and renders prorated probability for each scenario in that set" (Auguston 2018, 52).

Both Monte Carlo simulation and Monterey Phoenix models are executable. Unlike Monte Carlo simulation, Monterey Phoenix supports "automated and exhaustive …scenario generation for early system architecture verification" up to a user-defined scope limit (Auguston 2018, 5). As such, Monterey Phoenix can provide a more formal and complete set of process-behavior scenario outcomes that results in a larger sample size of scenarios using the same statistical analysis performed with historical project data. However, given the lack of extensive experience with Monterey Phoenix, we determined that the use of Monterey Phoenix was more appropriate for future work.

## D. APPLICABILITY TO EXISTING RESEARCH

There are a significant number of research efforts pertaining to process modeling and software process modeling. A search of software process modeling in scholarly journals and other peer-reviewed publications within the past 10 years produced 320 results. While not all of these may be applicable to research for this thesis, the search results demonstrate the breadth of research conducted or being conducted related to software process modeling. As part of this thesis, we reviewed 15 different sources and selected six research efforts that have or potentially have correlation with this thesis. These included the following research: An Agile Project System Dynamics Simulation Model (White

2014), A Reference Model for Simulating Agile Processes (De Silva, Rayadurgam, and Heimdahl (2015), Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods (Glaiel, Moulton, and Madnick, 2013), Modeling Dynamics in Agile Software Development (Cao, Ramesh, and Abdel-Hamid, 2010), Managerial Implications and Comparative Effects of SAFe Scaled Agile Methods in Government Software Acquisition (Moulton et al. 2017), Early Phase Cost Models for Agile Software Processes in the U.S. DoD (Rosa et al. 2017, 34).

The research conducted by White (2014) focused on helping project managers and their ability to more accurately forecast agile development process performance versus waterfall for NASA. While our thesis does not directly compare agile and waterfall software development modeling processes, White's research provides greater insight into the development cost for agile software development (2014, 74). In addition, White's research included analysis and comparison of rework required between agile and waterfall methods. The process architecture for this thesis captures rework cycles within the model and is accounted for in the IDEF0 and action diagrams. Our model does not currently quantify planned versus completed task as a function of rework. Further analysis and comparison of the research by White (2014) may provide insight into being able to quantify rework as part of the agile framework.

The research conducted by De Silva, Rayadurgam, and Heimdahl (2015) "introduces a process simulation reference model that provides the constructs and relationships for capturing the interactions among the individuals, product, process, and project in a holistic fashion—a necessary first step towards a process evaluation environment for agile processes" (De Silva, Rayadurgam, and Heimdahl 2015, 1). The author's reference model for simulating agile processes encompassed modeling behavior of individuals as well as decoupling product from process. An agent-based modeling approach was explored and resulted in "implementing models to represent individual behavior using agents as abstractions of people" (De Silva, Rayadurgam, and Heimdahl 2015, 90). Research for this thesis primarily focused process architecture modeling but it was not constructed with agent-based modeling methods and rather than decoupling product and process, they were modeled in a holistic and integrated approach.

The research conducted by Glaiel, Moulton, and Madnick (2013) focused on creating a framework for seven agile characteristics to explore how different combinations of agile features impact outcomes. These seven characteristics are defined as the "agile genome," or the core set of characteristics that are required to truly be agile (Glaiel, Moulton, and Madnick 2013, 3). Two of these core characteristics include customer involvement and continuous integration. It is important to note that Glaiel, Moulton, and Madnick's research was also done within the context of commercial and government organizations. The incorporation of continuous integration and customer involvement for agile development within a government organization aligns to the work in our thesis pertaining to the process decomposition of the SOPs into distinct actions that form together in a holistic fashion to produce a holistic architecture.

The research conducted by Cao, Ramesh, and Abdel-Hamid (2010) resulted in creation of a system dynamics model that accounts for interdependencies of agile development practices. As with the process architecture model for this thesis, structural and behavioral tests were extensively used by Cao, Ramesh, and Abdel-Hamid to validate the system dynamics model (Cao, Ramesh, and Abdel-Hamid 2010, 15). Cao, Ramesh, and Abdel-Hamid's research focused on enhancing "understanding of agile software development, especially the dynamic nature of agile practices when viewed as an integrated system" (Cao, Ramesh, and Abdel-Hamid 2010, 22). The scope of which included analysis of agile practices to include cost, schedule, and project scope. This included developer team size, sprint durations, and requirements volatility. Gaining further insight into how requirements volatility can be accounted for within our process architecture model is an area for further exploration.

The research conducted by Moulton et al. (2017) focused on scaled agile framework (SAFe), which is intended to be able to scale smaller teams of agile developers in order to meet larger organizational agile development efforts. While our research does not focus on scaled agile, the action diagrams developed are intended to help managers visualize and understand the application of the agile framework within the organization. Surprisingly, a search for scholarly journal articles in the past 10 years on scaled agile framework only provided 23 results. Scaled agile is still a relatively new field within agile; however,

furthering our research to scale the process architecture for an enterprise level requires a thorough understanding of how scaled agile should be implemented.

The research conducted by Dr. Giammarco's AMBIA dissertation (2012) provided insight to heuristics for architecture development. This work influenced how we developed the action diagrams and IDEF0 diagrams for the process architecture developed for this thesis. Further collaboration for the purposes of a process architecture for software development projects is applicable to the research for this thesis. Using the tenets of system architecture, software process modeling can be extrapolated to higher-level architecture diagrams and assist organizations in developing MBSE solutions that are able to address the interdependencies and complexities of agile software process architectures.

The research by Rosa et al. (2017) focused on early phase cost modeling within the U.S. DoD. The objective of this research was "to improve cost estimation by investigation available sizing measures, and providing practical effort estimation models for agile software development projects" (Rosa et al. 2017, 1). This research also provides insight into structured methods of gathering data, normalizing data for analysis, and measures to assess model validity. Variables included product and staff size. Data collection, analysis, measures of validity and staff size are directly related to the research for our thesis; however, product size and its impact on resulting cost estimations is an area for further exploration and collaboration. Additionally, Rosa et al. (2017) used a static cost modeling approach whereas our process architecture model uses effort-based simulation.

## E. MODELING APPROACH

The model types used within Innoslate are action and IDEF0 diagrams. The modeling approach includes the following: assessment of the SOPs, use of LML, architecture development conventions for action diagrams and IDEF0 diagrams, definition of system boundaries, simulation of the models, model testing, and verification and validation (V&V).

The action diagrams were constructed as a discrete model. "Discrete models contain distinct…entities that move through the process and can have attached attributes.

Change happens in discrete steps. This supports sophisticated, detailed analyses of the process and project performance" (Kellner, Madachy, and Raffo 1999, 15). Discrete modeling facilitated the successful completion of the model to ensure no errors were present. Innoslate supports Monte Carlo simulations in addition to discrete simulations. Monte Carlo is a stochastic simulation technique. "Stochastic modeling recognizes the inherent uncertainty in many parameters and relationships. Rather than using (deterministic) point estimates, stochastic variables are random numbers drawn from a specified probability distribution" (Kellner, Madachy, and Raffo 1999, 15). In addition to the core function of software development, there are other supporting functions modeled. These include business and technical feasibility analysis, contracts, and personnel assessment.

The essence of modeling is to provide a representation of systems in order to "predict and analyze performance, costs, schedules, and risks and to provide guidelines for systems research, development, design, manufacture, and management" (Maier 2009, 12). To model the architecture described within this thesis, we utilized the MBSE tool Innoslate version 3.9 to construct action diagrams and corresponding IDEF0 diagrams. We selected Innoslate as the MBSE process architecture tool for this thesis based on its ability to create action diagrams that can generate simulated output based on customizable metric-based parameters, the ability to generate corresponding IDEF0 diagrams that capture process functionality, and the fact that common assets that are used across separate diagrams have traceability throughout the architecture. Additionally, the fact that Innoslate projects are stored online facilitates collaboration among authors and advisors. The adoption and implementation of agile model-based system engineering can be thought of as a flowchart (see Figure 5).

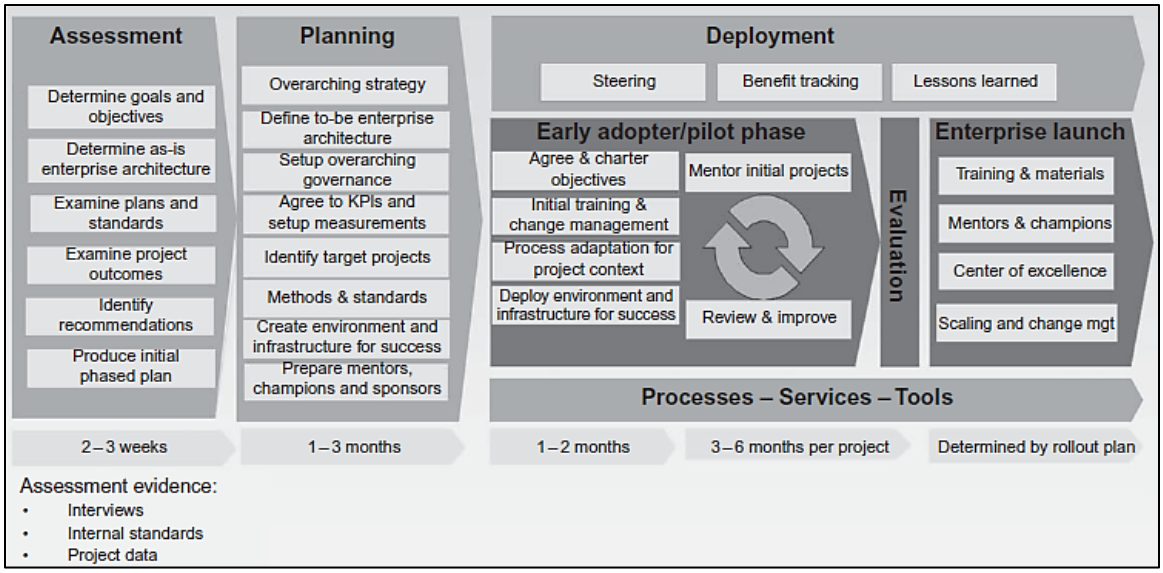



Figure 5. Incorporating Agile Model-Based Systems Engineering.
Source: Douglass (2016).

In the process of creating diagrams in Innoslate, an LML approach was utilized to capture the nuances of agile software development as captured in the organizational SOPs. According to the Innoslate developer, SPEC Innovations (2018, 1), Innoslate provides a description of its ontology and diagrams by utilizing the Systems Modeling Language (SysML) and LML and Life cyclemodeling.org's LML Specification 1.1 (2015, 3) which states:

> LML was designed with 6 major goals.
> 1. To be easy to understand,
> 2. To be easy to extend,
> 3. To support both functional and object oriented approaches within the same design,
> 4. To be a language that can be understood by most system stakeholders, not just Systems Engineers,
> 5. To support systems from cradle to grave,
> 6. To support both evolutionary and revolutionary changes to system plans and designs over the lifetime of a system.
>
> Most of the systems engineering community recognizes MBSE's ability to evolve, reuse and execute models is a significant improvement over the classic "document-based" approach's static view of a system. (Life cyclemodeling.org's LML Specification 1.1 2015, 3)

The adaptable and easily understood design of LML is beneficial in project management and systems engineering applications. Within the Innoslate project, several primary features of the tool were utilized. The features of Innoslate that help construct the system architecture each has its own nomenclature. At a high level, the project logically organizes information with a collection of items called entities. With respect to Innoslate process architecture, “an entity is something that can exist by itself and is uniquely identifiable” (SPEC Innovations 2018, 2). There are seven classes of entities that enable system design: requirement, artifact, action, asset, input/output, conduit, and characteristics. For the purposes of generating action diagrams within this project, modeling involved utilizing actions with assets, inputs, and outputs. Within action diagrams in this process architecture, an LML “action entity specifies the mechanism by which inputs are transformed into outputs” (Life cyclemodeling.org 2015, 11). An LML “asset entity specifies an object, person, or organization that performs Actions, such as a system, subsystem, component, or element” (Life cyclemodeling.org 2015, 11). An LML input or output (I/O) entity provides “the information, data, or object input to, trigger, or output from an Action” (Life cyclemodeling.org 2015, 11). An I/O provides a means to guide the flow of actions within Innoslate. Each entity can have attributes, which are inherent features that can also provide a means to quantify attributes such as cost or duration for an entity.

The specific teams that perform actions within each SOP were associated with the corresponding performing assets in Innoslate as part of the modeling process. In performing this function, it was imperative that we utilize common language to capture nomenclature for those performing actions so that there was consistency among the diagrams. Utilizing this approach ensures simplicity, consistency, and accountability for personnel responsible for performing functions. For example, one instantiation of the configuration management (CM) team was utilized throughout all models, rather than variations of CM group or CM representative. The process of building the architecture was an iterative process, which included decomposing higher-level assets to provide depth within the model that further defines the architecture within the boundary conditions.

When constructing diagrams in Innoslate, the four main SOPs were referenced, but additional insight was also obtained from subject matter experts (SME) within the C2IS division to ensure accuracy in the architecture model and its fidelity to their actual processes. To model the interdependencies and connections between processes described in the SOPs, we defined the functional architecture by using action diagrams. Based on the activities performed within the SOPs, actions were assigned to corresponding physical elements within the architecture. "The function name should start with an action verb and include an object of that action. The verb should not contain an objective or performance goal such as maximize, but should describe an action or activity that is to be performed" (Buede 2009, 204). Within action diagrams there are action blocks, and each has an entity which can be customized. Characteristics can be added to provide more detail regarding relationships and quantifiable metrics. These further define the architecture and provide users with a dynamic tool that can holistically represent the architecture. The architecture tools that are utilized to build action diagrams in Innoslate are pictured in Figure 6.

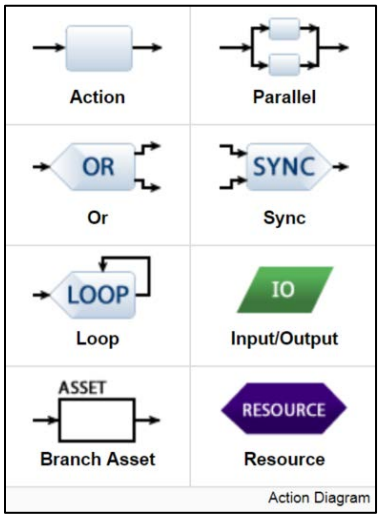



Figure 6. Innoslate Action Diagram Architecture Objects.
Source: SPEC Innovations (2017).

The physical implementation of the architecture entails association with physical elements, which can include software, hardware, or human elements. Physical elements within Innoslate are referred to as assets, and are denoted using branch labels as shown in

Figure 7, which shows the assets Team 1 and Team 2 performing two separate actions on separate branches.

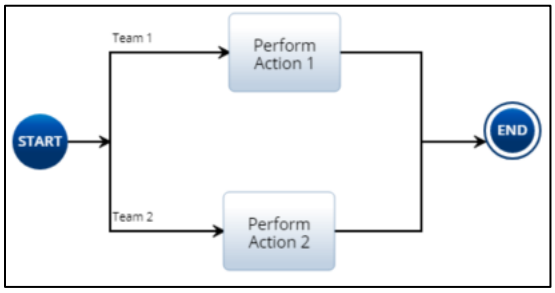



Figure 7. Innoslate Action Diagram Assets

Input and output (I/O) entities need to be used to communicate between actions in Innoslate. “In Innoslate, input / output entities are the primary form of communication between actions. In order for items to flow among the elements in our system, the components will need to communicate with each other through some type of connections” (SPEC Innovations 2018, 5). Innoslate performs this function through input/outputs, shown in Figure 8 using a green parallelogram. In addition, providing a common connection between actions, I/O entries control and guide the flow of processes within an Innoslate action diagram by triggering subsequent actions and creating a sequence of activities.

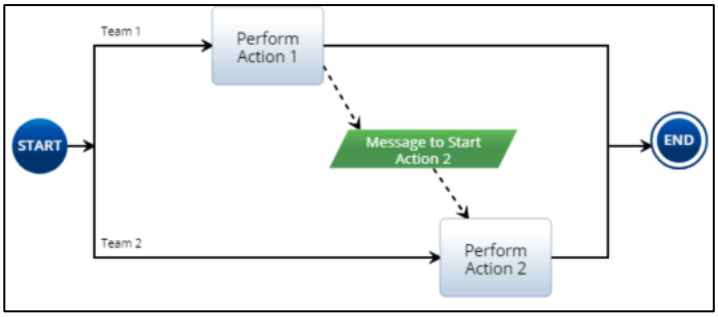



Figure 8. Innoslate Action Diagram I/O

Each diagram created within Innoslate linked with other diagrams and entities within the architecture to create relationships, which provide a means of traceability. This

builds off a feature within Innoslate that decomposes an Action diagram at a component level to show another layer of detail. The architecture decomposition process is analogous with defining characteristics of a forest from 10,000 feet, then from the treetops, then from the forest floor, and finally, from down in the weeds. Additional features within Innoslate action diagrams include the ability to build loops to show cases in which an action repeats. Loops can be set to repeat a pre-set number of times or pre-set with a probability of repeating. Figure 9 provides an example of an action diagram loop.

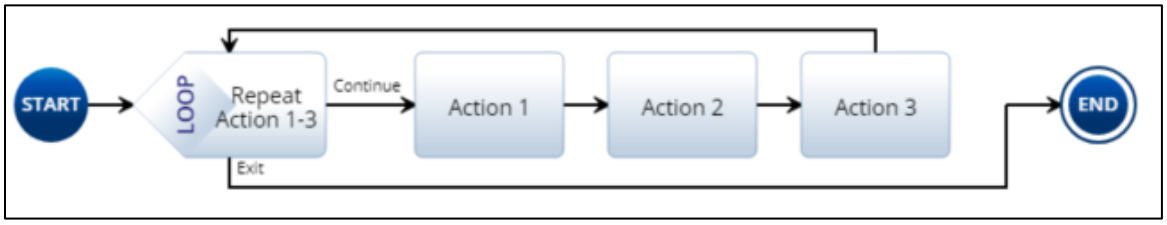



Figure 9. Innoslate Action Diagram Loop Function

Action diagrams also provide the opportunity to setup “OR” actions to provide an opportunity to select between two or more options in a sequence of activities. Each action on an “OR” action branch must pair 1-for-1 with corresponding actions on each branch of a “SYNC” action. The 1-for-1 pairing between “OR” and “SYNC” action is completed by utilizing I/O items to provide a physical item that provides linkage between the “OR” and “SYNC” action. When activities corresponding to an “OR” action are performed on separate branches of an action diagram, it is necessary have a “SYNC” action to ensure the model handles completion of the action appropriately. If an “OR” action is not correctly synchronized within the model, the action diagram will produce an error when simulated. Figure 10 provides an example of an “OR” action being synchronized between two team assets.

Action diagrams are created using combinations of these assets, entities, and actions within Innoslate, and generating action diagrams supports development of corresponding IDEF0 diagrams. The IDEF0 diagrams utilize the activities performed in action diagrams and graphically display the associated forms for the functional architecture. Another way

to consider this is that the activities performed in the action diagrams are expressed as functionality in the IDEF0 diagrams.

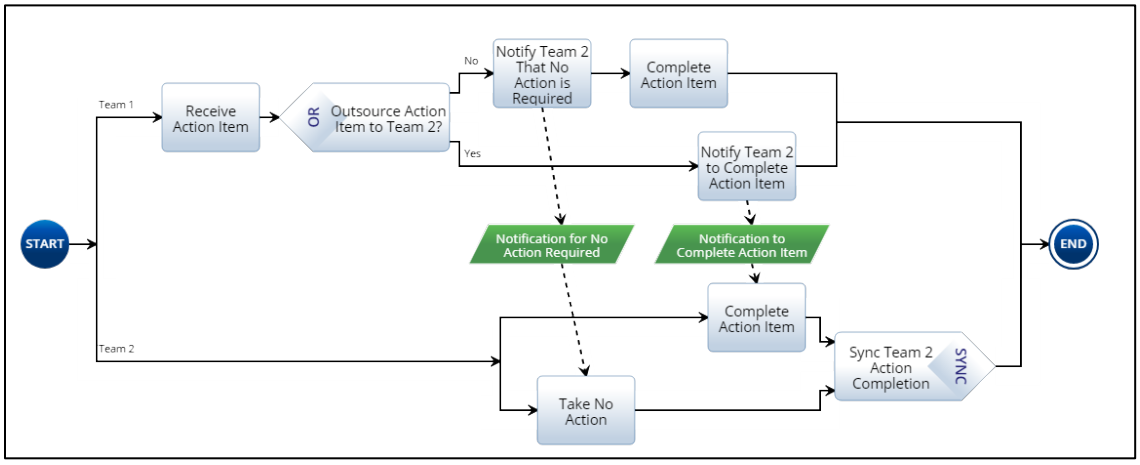



Figure 10. Innoslate Action Diagram OR Synchronization Function

According to Buede (2009, 66), an IDEF0 model provides a perspective that focuses on a system's functional or process model and describes it by utilizing a graphical modeling language with a holistic methodology for creating models. The IDEF0 model "answers definitive questions about the transformation of inputs into outputs"(Buede 2009, 67), and it provides context by establishing the system boundary. The architectural design process requires consideration of system boundaries in order to determine and make clear what is inside and outside those boundaries (Crawley, Cameron, and Selva 2017, 24). If required, the boundary conditions for the IDEF0 diagrams are further defined to provide the viewer with additional context. System boundaries help to divide entities within the system from accompanying and external entities (Crawley, Cameron, and Selva 2017, 123). The IDEF0 diagrams are utilized with the MBSE tool, Innoslate, to capture the SOP system boundaries within the holistic architecture. The IDEF0 has one viewpoint from the perspective of the topic system, which helps provides a common environment additional context for understanding the system. This type of diagram leverages an interconnected set of diagrams, which utilize the intuitive flow of graphics in conjunction with corresponding descriptive verbiage in the graphics. Decomposition of Innoslate IDEF0 diagrams provides

a greater level of detail for functions that are found at a higher level within the architecture. The top-level IDEF0 diagram "defines the inputs, controls, outputs, and mechanisms (ICOMs)" (Buede 2009, 67) for the subsequent decomposed diagrams. The context of an IDEF0 "establishes the boundaries of the system or organization being modeled by defining the inputs and controls entering from external systems and the outputs being produced for external systems" (Buede 2009, 67). For this thesis, the IDEF0 diagrams were created using the Innoslate architecture entities shown in Figure 11. These included basic mechanisms, inputs, outputs, and controls.

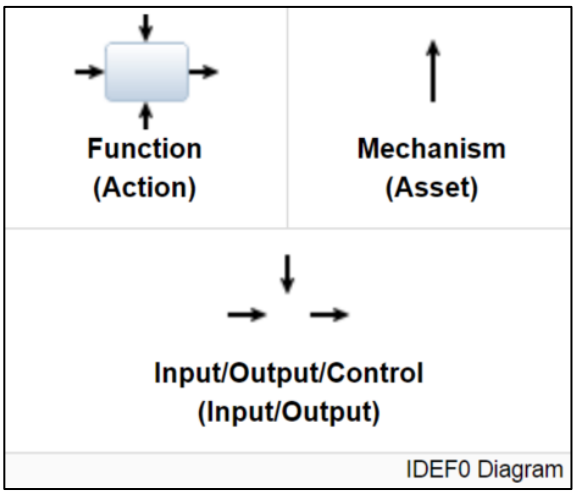



Figure 11. Innoslate IDEF0 Diagram Architecture Tool.
Source: SPEC Innovations (2017).

An additional feature within Innoslate is the "Entity View," which is associated with each asset block in an architecture model. The "Entity View" is accessible for any given asset or collection of assets by selecting it from a drop-down menu. An image of the "Entity View" graphical user interface (GUI) is shown in Figure 12. This view enables the user to apply metrics such as duration and cost to specific assets within the architecture. It is through modification of the "Entity View" metrics that assets in Innoslate can be accurately quantified to yield realistic simulation output for a given set of assets. Additionally, the "Entity View" provides a method for managing relationships and linkages between assets to ensure traceability.

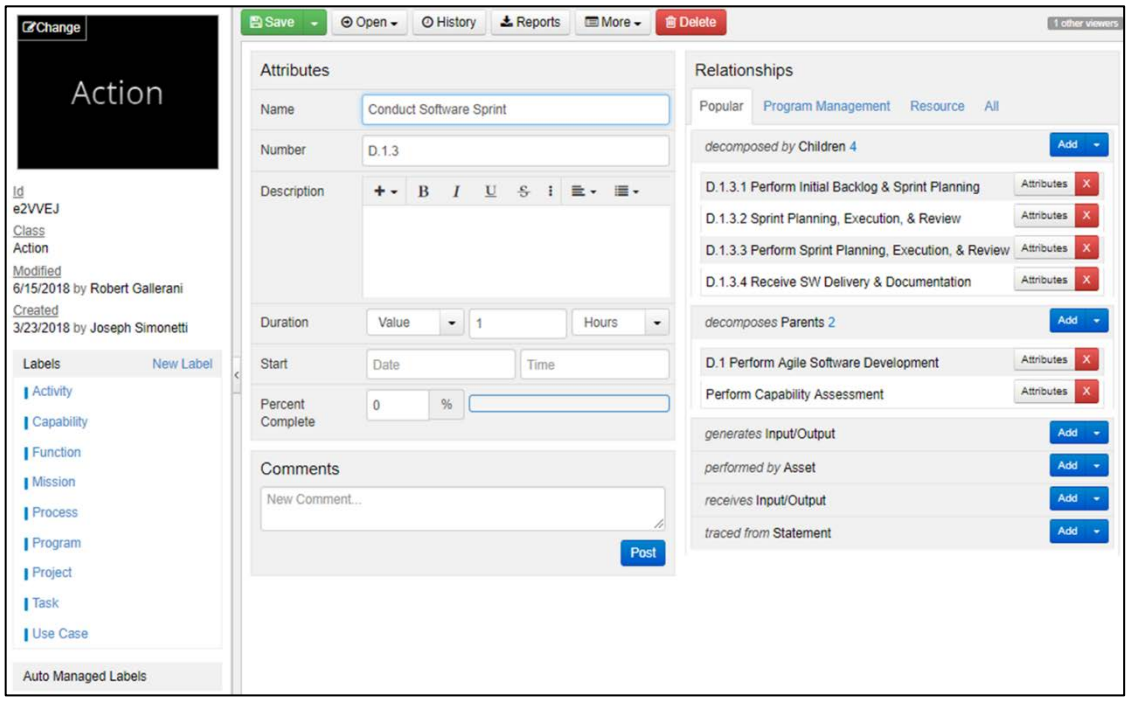



Figure 12. Innoslate Entity View

## F. CHAPTER SUMMARY

This chapter provided a brief literature review of similar research in the context of agile software development modeling and simulation. Software process models, to include waterfall, spiral, incremental, V-model, and the agile development process were discussed. In the context of this thesis, agile software development is used exclusively. A detailed discussion of the MBSE tool, Innoslate, was provided for insight into how the process architecture was developed for this thesis. The overview of Innoslate and process models provides the reader with appropriate context in order to have a better understanding of the models presented in Chapter III and Chapter IV.

# III. MODEL DEVELOPMENT AND RESULTING ARCHITECTURE

## A. INTRODUCTION

This chapter discusses the action and IDEF0 diagrams developed as part of an analysis of existing SOPs for agile software development. The scope of the action and IDEF0 models also includes business and technical feasibility analysis of candidate software capabilities, assessing and selecting existing personnel for the new work, and contractual actions to support technical activities. Discussion encompasses model purpose, decomposition of action diagrams to the lowest level required, IDEF0 diagrams to represent an architectural view and high-level representation of the action diagrams, model constraints, and boundaries. Models were vetted and validated through appropriate subject matter experts within the organization. Section B of Chapter III discusses the level 0 action and IDEF0 models and provides a high-level overview that serves as a foundation for the other diagrams that follow. Chapter III Section C discusses organizational processes models for business and technical feasibility of candidate products or deliveries. Chapter III Section D provides a model for assessment of available personnel to assign to new or existing work. Chapter III Section E discusses models for contracts planning, development, and execution within the organization. Chapter III Section F discusses models for agile software development. This chapter closes with Chapter III Section G, which provides conclusions and recommendations for the next actions to expand the modeling work done as part of this project.

## B. LEVEL 0 ACTION AND ARCHITECTURE MODELS

This section discusses the high-level action and IDEF0 diagrams that provides context for the major components of the architecture. The top-level action diagram is partitioned into four major actions: “business and technical feasibility analysis (B.1),” “assess available personnel (P.1),” “perform contracts development (C.2),” and “perform agile software development (D.1)” (see Figure 13). The action of “reject or redirect (R.1)” is a supporting action placed in parallel with the entire model and designed to handle reject

or redirect triggers from actions within the decomposed diagrams to end the process without further work. The actions “business and technical feasibility analysis (B.1)” and “assess available personnel (P.1)” are performed in parallel before “perform contracts development (C.2)” and “perform agile software development (D.1).” The primary rationale for this is to perform an initial assessment of the business and technical feasibility and perform an internal assessment of personnel, prior to pursuing contracting actions. Business and technical feasibility analysis focuses on gathering customer needs, understanding those needs, and ensuring the potential work properly aligns with the organizations mission and purpose. Assessing personnel is needed to determine which employees within the organizations would make the best fit for new opportunities that become available. A more detailed model decomposition and discussion of the business and personnel processes is provided in Chapter III Sections B and C.

Within the C2IS division, contracts development pertains to task orders issued under a multiple award contract (MAC) as described in Chapter I Section D. In a single award environment, one company is awarded all work and is the sole prime vendor. In a multiple award environment, there can be any number of companies that compete and are awarded the contract as a prime. As stated previously, it is beyond the scope of this thesis to model either the single award or multiple award contracting processes. The action models and resulting architecture are restricted to the task order process only. The actions “perform contracts development (C.2)” and “perform agile software development (D.1)” are performed in parallel because the work will be done by a team of government and contractors, or government workers without contractors. It is not preferable for the C2IS division to only act as a pass-through from the sponsoring agency to the contracting company. The “contractor workforce required (C.1)” “OR” function provides the option of performing work with or without a contractor workforce (see Figure 13). If contract task orders are needed, constructing the model with contracting and software development efforts in parallel facilitates partial work initiation while contracting actions are taking place. The entire high-level action diagram resides within a “SYNC,” which allows for some action paths to be taken while others are ignored within the decomposed diagrams for each major action.

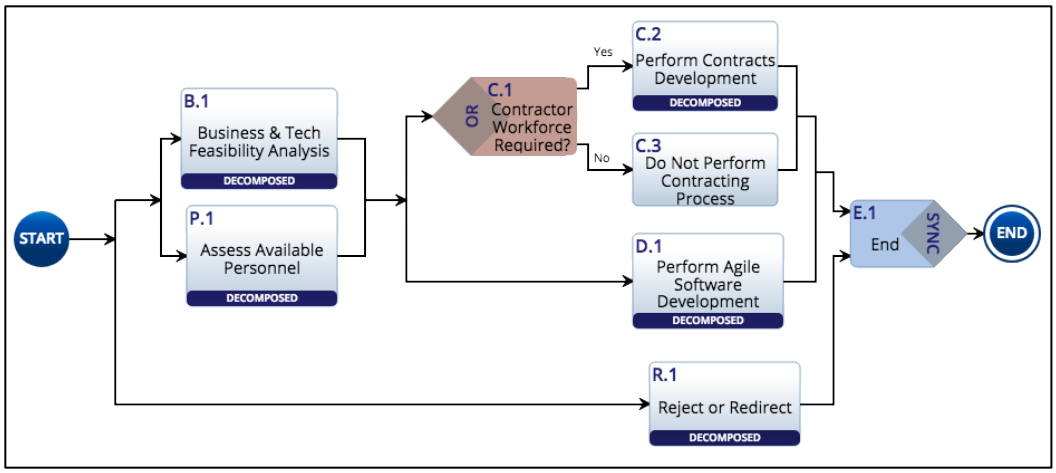



Figure 13. Top-Level Action Diagram

The top-level action diagram and all the decomposed diagrams for each major action were used to develop the architecture to enable agile software development (see Figure 14). The IDEF0 is the highest level architectural diagram. For traceability, “business and technical feasibility analysis (B.1)” in the top-level action diagram aligns to “perform business analysis (EXT.F.1)” in the IDEF0. The action “assess available personnel (P.1)” in the top-level action diagram maps to “perform organizational assessment of personnel (EXT.F.3)” in the IDEF0. In addition, “perform contracts development (C.2)” aligns to “develop and administer contract task orders for software development (EXT.F.2).” Lastly, “perform agile software development (D.1)” maps to “perform agile software development (F.0).”

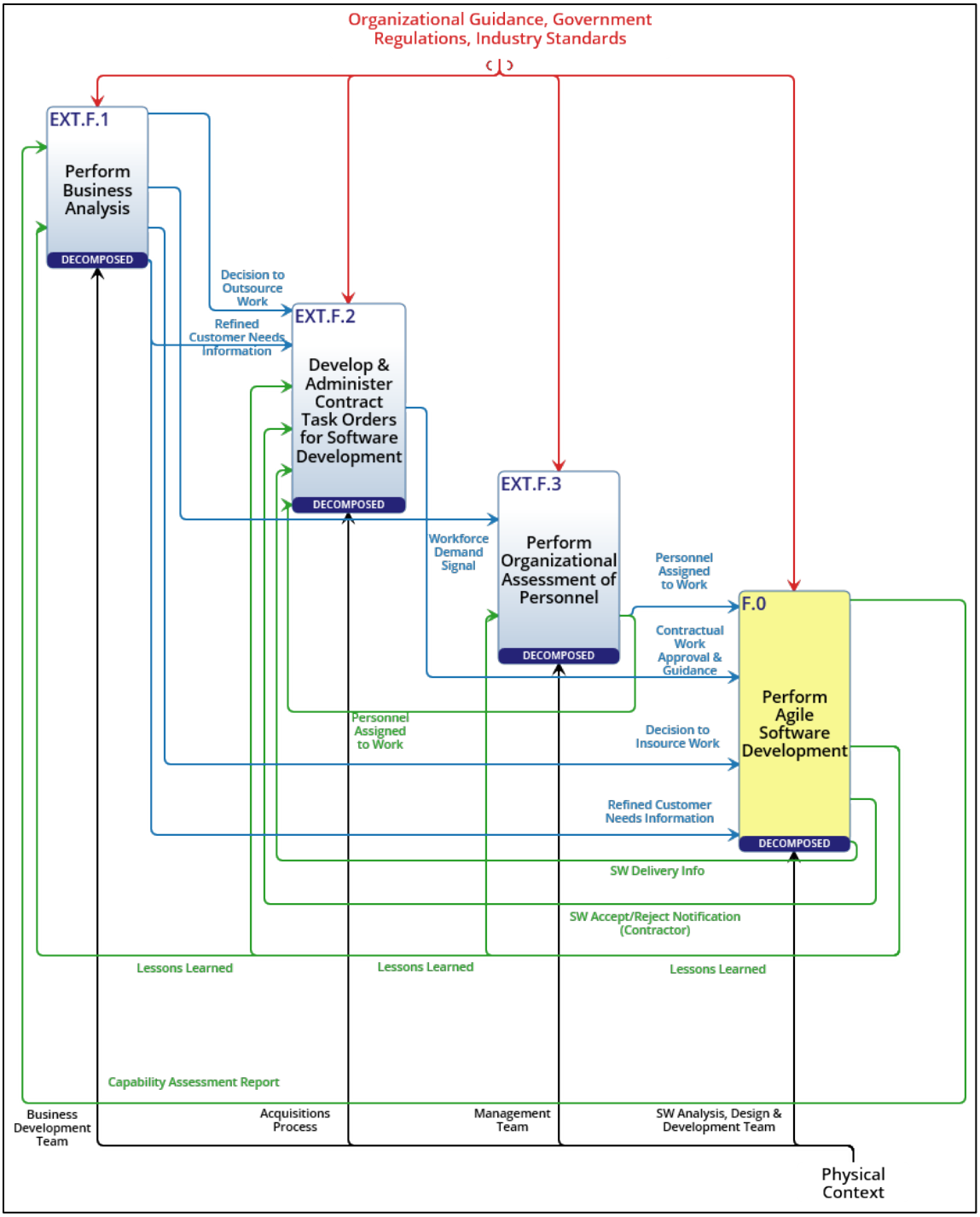



Figure 14. Architecture to Enable Agile Software Development IDEF0

The “business development team (EXT.C.1)” is the mechanism for “perform business analysis (EXT.F.1).” The “project acquisitions team (EXT.C.2)” is the mechanism for “develop and administer contract task orders for software development (EXT.F.2).” The function, “Perform organizational assessment of personnel (EXT.F.3)” is performed by the “management team (EXT.C.3).” The asset “software analysis, design and development team (C.0)” is the mechanism for “perform agile software development (F.0)” (see Figure 14).

Each of these functions receives a control for organizational guidance, government regulations, and industry standards, all of which can influence the ways and means that each function is performed (see Figure 14). For example, organizational guidance may include direction to avoid specific types of work, or to pursue specific types of work actively. Government regulations can influence how the C2IS division is able to execute contracting actions. Examples include periodicity of a contract task orders, requirements for small business set-asides, or other regulatory requirements. Industry standards include best practices or other generally accepted practices and technical standards for software development.

The purpose of “perform business analysis (EXT.F.1)” aligns to that of B.1 in the top-level action diagram (see Figure 13). Business analysis focuses on gathering customer needs, understanding those needs, and ensuring the potential work properly aligns with the organizations mission and purpose. Prospective work can be handled via insourcing, outsourcing, or a combination of both. The function “perform business analysis (EXT.F.1)” provides the decision to insource work as an output and is carried down as an input of “perform continuous software development and integration (F.0).” Similarly, the decision to outsource work is provided from EXT.F.1 to “develop and administer contract task orders for software development (EXT.F.2).” Refined customer needs are provided as an output from “perform business analysis (EXT.F.1)” to both “develop contract task orders for software development (EXT.F.2)” and “perform agile software development (F.0).” The last output from “perform business analysis (EXT.F.1)” is a technical demand signal provided as an input to “perform organizational assessment of personnel (EXT.F.3)” (see Figure 14). Lessons learned are provided as an input to “perform business analysis

(EXT.F.1)" from "perform agile software development (F.0)." Additionally, if a software capability is part of a larger existing system or new science and technology effort, then it will undergo a capability assessment to determine whether the capability should be accepted for further work. This capability assessment report is provided for consideration in the decision to accept, reject, or redirect work. The details of accepting, rejecting, or redirecting work is discussed in the decomposition of "perform work acceptance process (B.1.2)" and "perform business analysis (EXT.F.1)" (see Figure 17 and Figure 19).

The purpose of "develop and administer contract task orders for software development (EXT.F.2)" is to develop, compete, award, and monitor task orders that are used to supplement the government workforce. This function aligns to "perform contracts development (C.2)" in the top-level action diagram (see Figure 13). A task order can be competed under an applicable existing multiple award contract vehicle within the organization once the decision to outsource work and customer needs are captured. The subsequent award results in contractual work approval and guidance as an output from "develop and administer contract task orders for software development (EXT.F.2)" to "perform agile software development (F.0)" (see Figure 14).

The purpose of "perform organizational assessment of personnel (EXT.F.3)" is aligned to action P.1 in the top-level action diagram (see Figure 13). Assessing personnel determines which employees within the organization would make the best fit for available or future work. Once the technical demand signal is received and a personnel analysis is performed, the management team will assign personnel to work. This is done via an output from EXT.F.3 to both "develop and administer contract task orders for software development (EXT.F.2)" and "perform agile software development (F.0)." Technical workers are needed to help with activities performed for developing and accessing contract task orders, and other personnel will be assigned to be a part of the software analysis, design, and development team (see Figure 14).

"Perform agile software development (F.0)" is the main top-level function within the architecture. The function F.0 takes place after all applicable business and personnel analysis are complete, and contracting actions are at least started. Inputs to F.0 were discussed previously as part of the other functions in the top-level diagram. Outputs from

F.0 include lessons learned from software development activities that are provided as a feedback loop and input to all other functions. Software delivery information is provided as an input to "develop and administer contract task orders for software development (EXT.F.2)." This is to help the technical members of the project acquisitions team determine how to write the task order in a way that specifies what the government expects and how software deliveries are to be done. The other output from F.0 is a notification to accept or reject, which is provided to the contractor if the software delivery is rejected for any reason. This output is provided as an input to "develop and administer contract task orders for software development (EXT.F.2)" (see Figure 14).

## C. BUSINESS AND TECHNICAL FEASIBILITY ANALYSIS (B.1)

This section discusses the detailed decomposition of the "business and technical feasibility analysis (B.1)" action diagram and the "perform business analysis (EXT.F.1)" IDEF0. Within the action diagram for B.1, there is an option to reject or redirect work, which provides a trigger into the top-level function of "reject or redirect (R.1)." As such, the action diagram for reject or redirect is described in this section to provide context for its functionality within the process architecture. The action "business and technical feasibility analysis (B.1)" decomposes into two actions: "receive and analyze customer needs (B.1.1)," and "perform work acceptance process (B.1.2)" (see Figure 15). As previously discussed, the purpose of this model is to gather customer needs, analyze and understand those needs, and apply a process for work acceptance to ensure the organization is taking on appropriate work.

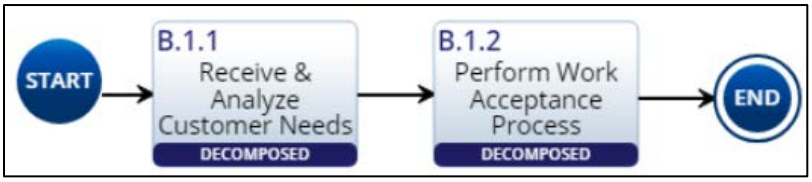



Figure 15. Decomposition of Business and Technical Feasibility Analysis (B.1)

### 1. Receive and Analyze Customer Needs (B.1.1)

The decomposition of “receive and analyze customer needs (B.1.1)” is partitioned into three parallel paths, each performed by a unique asset (see Figure 16). These assets include organizational leadership, the general workforce, and project managers. There is a wide variety of personnel who may receive customer demand signals. Within the C2IS division, personnel are not permitted to commit to new work without vetting the opportunity through organizational leadership. Leadership must ensure that the organization is performing the right work, it does not conflict with other efforts, and it aligns with the C2IS division’s mission. This is reflected in the diagram as customer demands, which are received by anyone and reported up to organizational leadership (see Figure 16). Once leadership receives a demand signal, guidance will be given to a project manager to perform a customer needs analysis. The project manager will lead a team of people in the customer needs analysis. This is often an iterative process until the customer needs are fully understood. After the customer needs are understood, the project manager will provide recommendations to leadership on the technical approach and then leadership will provide direction to a project manager to initiate the work acceptance process. Even if the work is ultimately not accepted, the work acceptance process is used to capture redirected or rejected work in addition to accepted work.

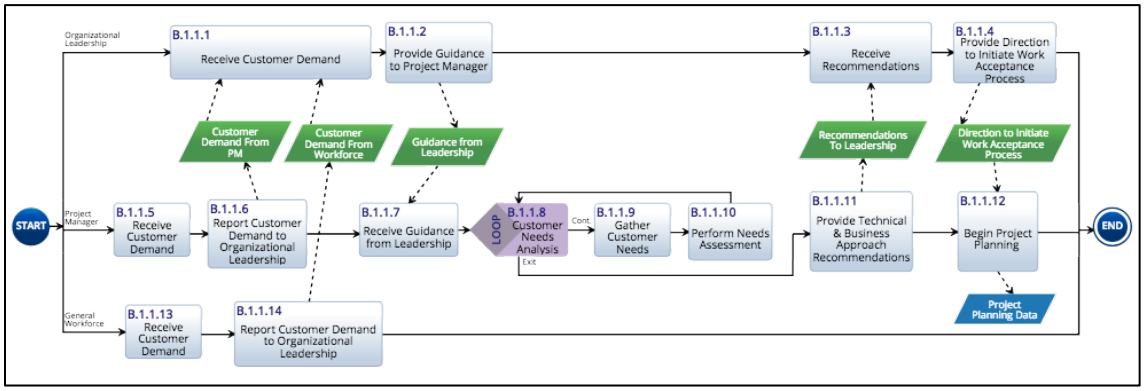



Figure 16. Decomposition of Receive and Analyze Customer Needs (B.1.1)

### 2. Perform Work Acceptance Process (B.1.2)

The action “perform work acceptance process (B.1.2)” is performed in parallel by organizational leadership and project managers. Initially, the project manager performs project planning efforts (see Figure 17). Project planning will gather and provide current and future year funding information, such as funding amounts, appropriations, and sponsoring organizations. Project planning includes statements of work to be performed, anticipated number of government and contractor employees, a high-level risk assessment, anticipated material or services procurements, and information regarding any external agency agreements that may be required. Organizational leadership will use this project planning data to determine if the work should be accepted, rejected, or redirected. If leadership determines the work is rejected, they will notify the project manager who will provide the rationale for rejecting the work to the prospective customer. The work may be redirected as well, meaning that there may be a recommendation for the work to be done by another entity within the same organization, or redirected to another external organization. If the work is accepted, leadership must make and disseminate the decision whether work with be insourced outsourced or a combination of both. The project manager will then begin executing the project plan based on direction from leadership. The action “begin executing project plan (B.1.2.11)” provides a workforce demand signal as an input to “perform personnel qualification analysis (P.1.1.1)” to initiate the process to find and assign personnel to the new work. Chapter III Section D discusses the details of the personnel assessment actions.

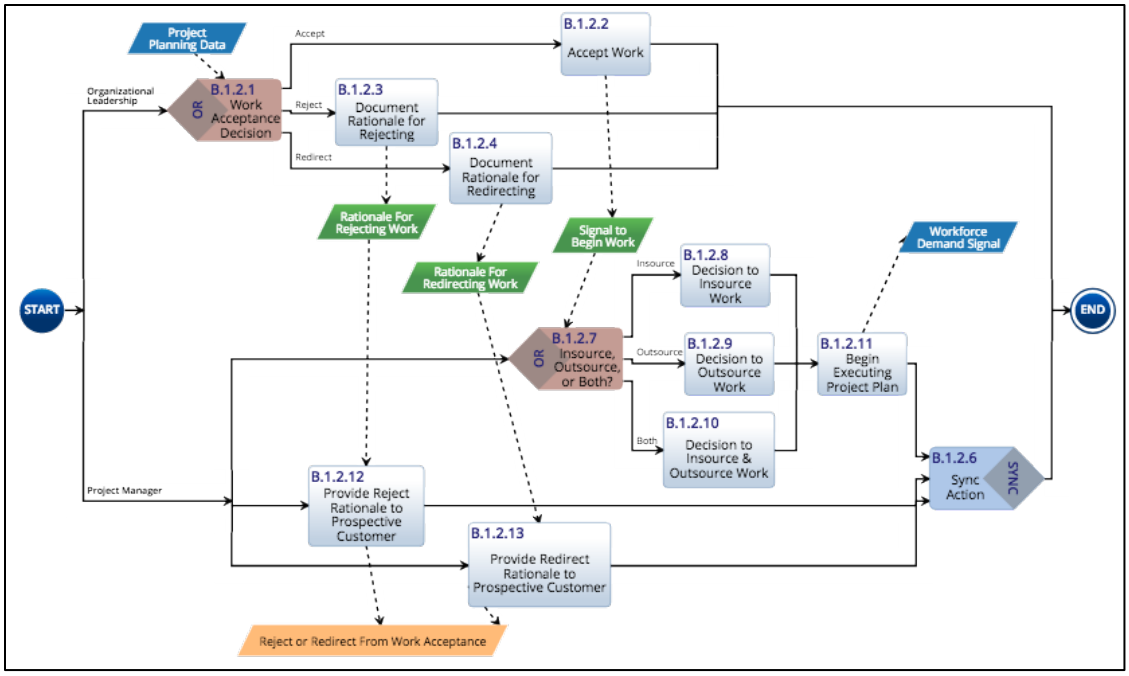



Figure 17. Decomposition of Perform Work Acceptance Process (B.1.2)

### 3. Reject or Redirect (R.1)

If the work is rejected or redirected, a trigger is provided to the top-level "reject or redirect (R.1)." The decomposed view of "reject or redirect (R.1)" receives all triggers to reject or redirect work throughout the entire model (see Figure 18). This action will end all activities and result in the model completing without performing other actions. The reject or redirect trigger is used extensively throughout the model; therefore, any follow-on discussion regarding reject or redirect will reference Figure 18.

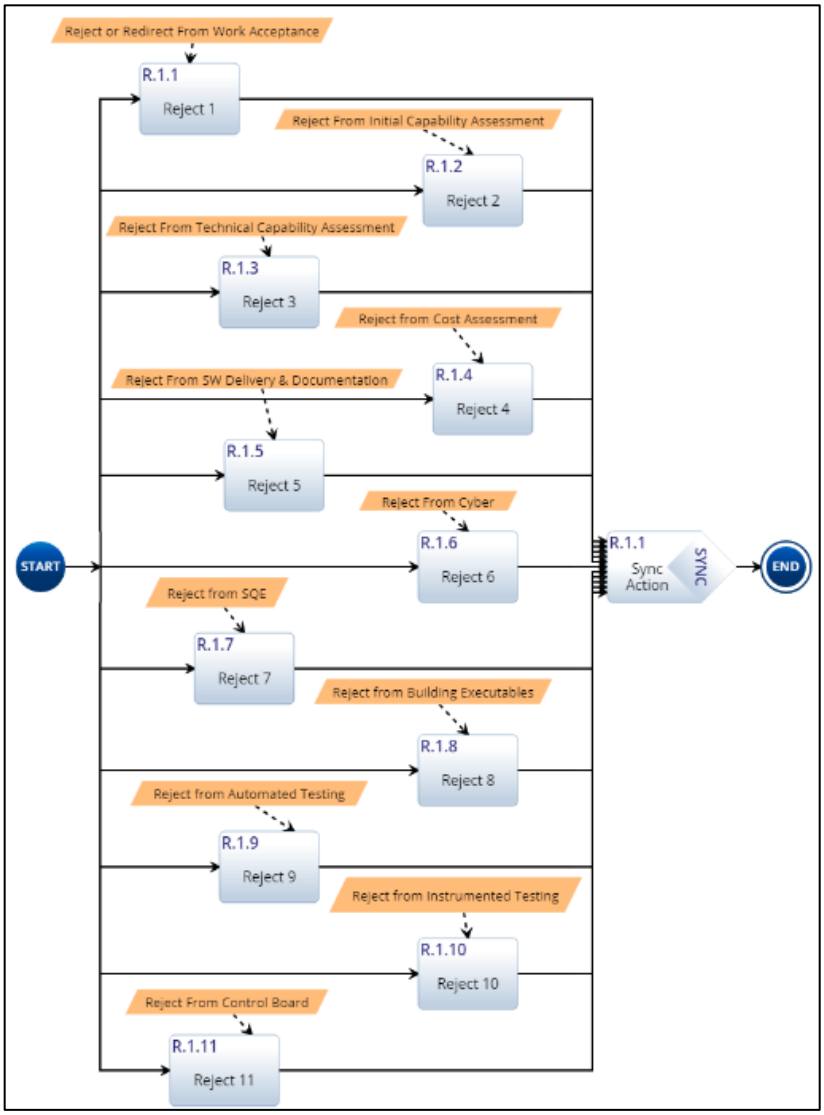



Figure 18. Decomposition of Reject or Redirect (R.1)

### 4. Perform Business Analysis (EXT.F.1)

The actions “receive and analyze customer needs (B.1.1)” and “perform work acceptance process (B.1.2)” were used to develop the decomposed architecture diagram of “perform business analysis (EXT.F.1).” As previously discussed, the purpose of this model is to gather and analyze customer needs, and apply a process for work acceptance. There are three main functions and mechanisms within the decomposition. The “organizational workforce (EXT.C.1.1)” is the mechanism for “receive customer demand (EXT.F.1.1).” The “project manager (EXT.C.1.2)” is the mechanism for “conduct needs analysis and

project planning (EXT.F.1.2).” “Organizational leadership (EXT.C.1.3)” is the mechanism for “accept, redirect, or reject work (EXT.F.1.3)” (see Figure 19).

The functions “receive customer demand (EXT.F.1.1)” and “conduct needs analysis and project planning (EXT.F.1.2)” align to “receive and analyze customer needs (B.1.1)” in the action diagram. The function “accept, redirect, or reject work (EXT.F.1.3)” aligns to “perform work acceptance process (B.1.2)” (see Figure 15). The organizational guidance, government regulations, and industry standards are carried down as controls from the parent diagram “perform business Analysis (EXT.F.1).” Customer needs can be received by anyone in the workforce. After capturing this information, preliminary customer needs information is provided from “receive customer demand (EXT.F.1.1)” as an input to “conduct needs analysis and project planning (EXT.F.1.2).”

The project manager will lead a team through initial efforts to perform a customer needs analysis and provide technical and business approach recommendations to leadership as an input to “accept, redirect, or reject work (EXT.F.1.3).” Organizational leadership will direct the project manager to initiate the work acceptance process. The project manager will then perform project planning and submit project-planning data to organizational leadership as an input to EXT.F.1.3. If organizational leadership decides to proceed, a determination is made whether to insource work, outsource work, or do a combination of both. If work will be outsourced, “accept, redirect, or reject work (EXT.F.1.3)” provides a decision to outsource work output that is carried down as an input to “develop and administer contract task orders for software development (EXT.F.2).” If the work will be insourced, “accept, redirect, or reject work (EXT.F.1.3)” provides a decision to insource work output that is carried down as an input to “perform agile software development (F.0).” The function “conduct needs analysis and project planning (EXT.F.1.2)” provides refined customer needs as an output that is carried down as an input to both “develop and administer contract task orders for software development (EXT.F.2)” and “perform agile software development (F.0).” The function “conduct needs analysis and project planning (EXT.F.1.2)” generates a workforce demand signal output that is provided as an input to “perform personnel qualification analysis (EXT.F.3.1)” (see Figure 19 and Figure 23). The function “accept, redirect, or reject work (EXT.F.1.3)” also receives a capability

assessment input from “perform agile software development (F.0).” This capability assessment is discussed in detail in Section F, Agile Software Development. Lastly, lessons learned is provided as an input to all functions within the decomposed “perform business analysis (EXT.F.1)” IDEF0. These lessons learned from “perform agile software development (F.0)” are used to refine product and process improvement continually.

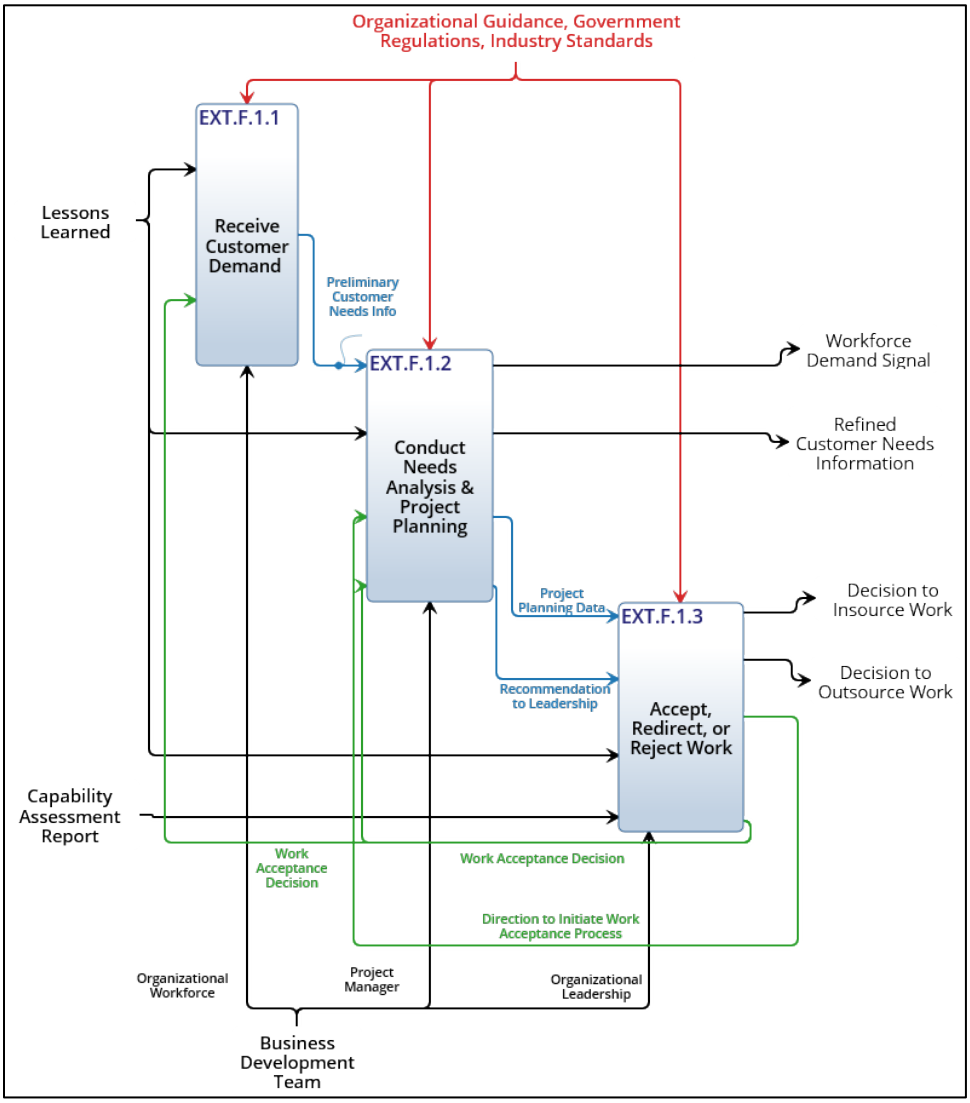



Figure 19. Architecture IDEF0 of Perform Business Analysis (EXT.F.1)

## D. ASSESS AVAILABLE PERSONNEL (P.1)

This section discusses the decomposition of the "assess available personnel (P.1)" action diagram and the "perform organizational assessment of personnel (EXT.F.3)" IDEF0. The action "assess available personnel (P.1)" decomposes into: "perform initial personnel assessment (P.1.1)" and "perform personnel selection (P.1.2)" (see Figure 20). As previously discussed, the actions for assessing personnel are used to determine which employees within the organization would make the best fit for available work.

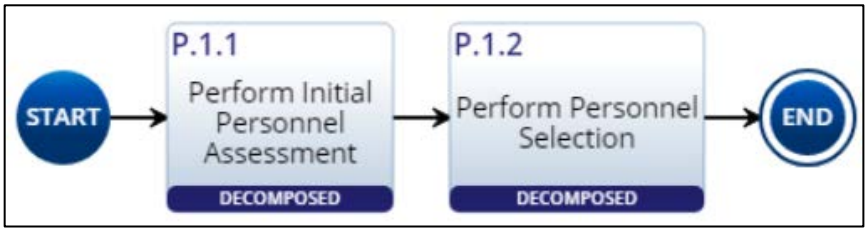



Figure 20. Decomposition of Assess Available Personnel (P.1)

### 1. Perform Initial Personnel Assessment (P.1.1)

Employees are aligned with supervisors who have administrative authority. The supervisor will "perform initial personnel assessment (P.1.1)." For example, a supervisor can direct employees to change projects, can approve time off, and can address disciplinary issues. A project manager directs day-to-day project activities, but does not have the same authorities as a supervisor does. The demand signal typically comes from a project manager to a supervisor in the form of technical skillsets required. Demand signals are vetted through supervisors because of the authority they have to direct employees from one project to another. It is best practice to socialize opportunities with the gaining and losing project manager, any supervisors involved, and the employee(s) affected; however, the final determination to move an employee from one project to another rests within the supervisory hierarchy.

Once the initial demand signal and technical skills thought to be required are passed along to a supervisor, he or she will perform an analysis of employee qualifications of personnel assigned under his or her supervision. The supervisor will down-select internal

candidates and provide information about the employees to the project manager for further consideration (see Figure 21). It is important to note that there are often many supervisors performing this analysis, and it can also be done across myriad technical competencies within the organization to provide a pool of potential candidates to project managers.

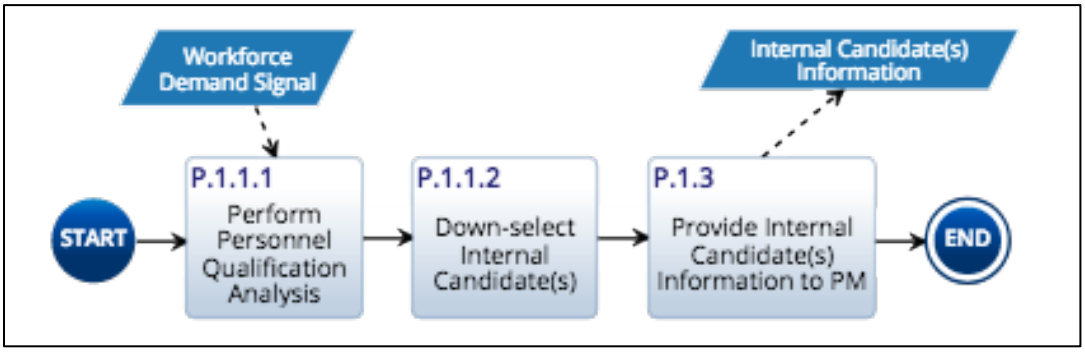



Figure 21. Decomposition of Perform Initial Personnel Assessment (P.1.1)

### 2. Perform Personnel Selection (P.1.2)

The supervisor will receive information about internal candidates from "perform initial personnel assessment (P.1.1)" and provide this information to the project manager for use in "perform personnel selection (P.1.2)." The project manager and supervisor execute the action to "perform personnel selection (P.1.2)" (see Figure 22). The same iterative question and clarification process between the supervisor and the project manager in "perform initial personnel assessment (P.1.1)" takes place in "perform personnel selection (P.1.2)." The project manager will discuss candidates with one or more supervisors until he or she determines the right mix of skill sets available among the candidates provided. The project manager will then make recommendations to the supervisor as to which personnel he or she request be assigned to the work. It is the supervisor's responsibility to discuss the new opportunity with their employees and make the staffing decision to assign appropriate employees and have them report to the project manager for work assignments (see Figure 22).

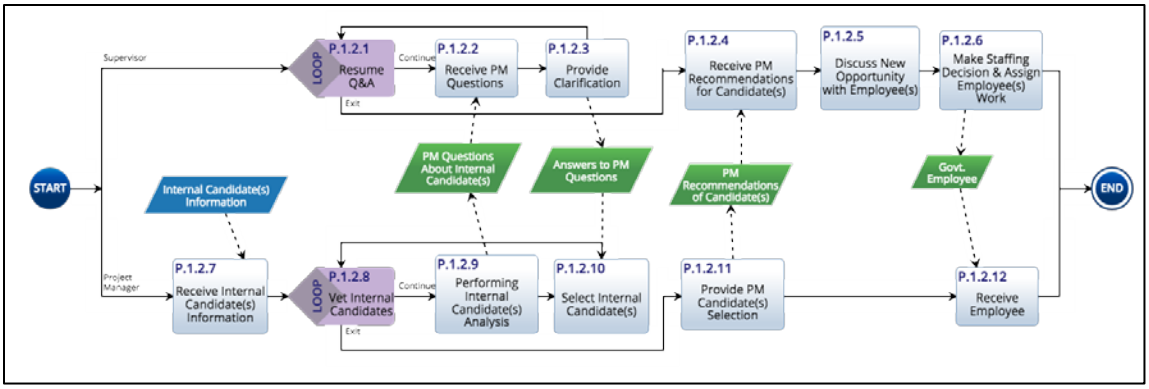



Figure 22. Decomposition of Perform Personnel Selection (P.1.2)

### 3. Perform Organizational Assessment of Personnel (EXT.F.3)

The decomposition of "perform initial personnel assessment (P.1.1)" and "perform personnel selection (P.1.2)" are used to develop the architecture IDEF0 for "perform organizational assessment of personnel (EXT.F.3)" (see Figure 23). As previously discussed, the actions for assessing personnel are used to determine which employees within the organization would make the best fit for available work. There are two main functions and mechanisms within the decomposition. The "supervisor (EXT.C.3.1)" is the mechanism for "perform personnel qualification analysis (EXT.F.3.1)." The "project manager (EXT.C.3.2)" is the mechanism for "interview and select candidate for task (EXT.F.3.2)" (see Figure 23).

The function "perform personnel qualification analysis (EXT.F.3.1)" aligns to "perform initial personnel assessment (P.1.1)" in the action diagram, and the function "interview and select candidate for task (EXT.F.3.2)" aligns to "perform personnel selection (P.1.2)." As with the IDEF0 for business analysis, a control in the form of organizational guidance, government regulations, and industry standards is carried down from the parent diagram EXT.F.3. Lessons learned from "perform agile software development (F.0)" are also provided as an input to both functions within the IDEF0 in order to refine the process for selecting technical personnel continually. The workforce demand signal generated from the business development team within "perform business analysis (EXT.F.1)" is provided as an input to "perform personnel qualification analysis (EXT.F.3.1)." As with the action diagrams, there is an information exchange between the

supervisor and project manager regarding employees being considered for work. Initial candidate information is provided as an output from "perform personnel qualification analysis (EXT.F.3.1)" to the function of "interview and select candidate for task (EXT.F.3.2)." The project manager may have questions about the candidates provided. This is reflected in the IDEF0 as an output called questions about candidates from EXT.F.3.2 back to EXT.F.3.1 as an input. Clarification regarding candidates is provided back to the project manager as an input. This process will continue until one or more personnel are selected for the work, shown as an output from "perform personnel qualification analysis (EXT.F.3.1)" to the input of "interview and select candidate for task (EXT.F.3.2)." The process ends with personnel assigned to work as an output of EXT.F.3.2.

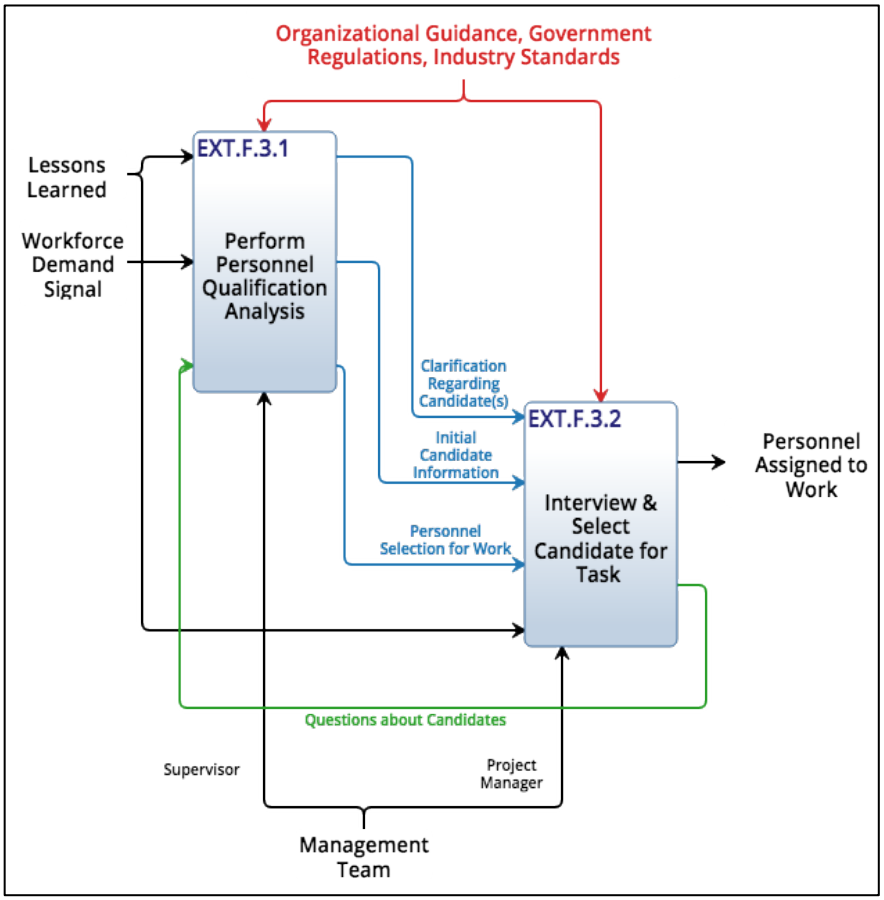



Figure 23. Architecture IDEF0 of Perform Organizational Assessment of Personnel (EXT.F.3)

## E. CONTRACTS DEVELOPMENT (C.2)

This section discusses the "perform contracts development (C.2)" action diagram and the "develop and administer contract task orders for software development (EXT.F.2)" IDEF0 decomposition. The action "perform contracts development (C.2)" decomposes into five actions: "perform market research (C.2.1)," "perform preliminary contracts planning (C.2.2)," "perform draft RFP activities (C.2.3)," "perform final RFP solicitation and award (C.2.4)," and "award task order and conduct COR activities (C.2.5)" (see Figure 24). The intent of the contracting actions is to develop, compete, award, and monitor task orders that are used to supplement the government workforce. As discussed in Chapter I Section D, the contracts processes within this model pertain to task orders awarded under an existing multiple award contract vehicle within the organization. The model does not account for the broad based contracting processes for either a single award or multiple award contract as both are not within the scope of this thesis. For context, the asset or mechanism of contracts pertains to a part of the organization that is warranted and sanctioned via the FAR to perform contracting operations and commitments on behalf of the government.


Figure 24. Decomposition of Perform Contracts Development (C.2)

### 1. Perform Market Research (C.2.1)

One of the first steps in the contracting process is to perform market research. The Federal Acquisition Regulations (FAR) defines market research as the collection and analysis of information such as product and supplier capabilities, and product characteristics to meet an agency's needs (FAR 2016, 275). The decomposition of "perform market research (C.2.1)" was partitioned into three parallel paths, performed by the project team, contracts personnel, and contractor organizations assets (see Figure 25).

To begin, the project team develops a draft statement of work (SOW) and request for information (RFI). These documents will be processed by the contracts team and ultimately provided to contractor organizations for review (see Figure 25). The SOW informs contractors of what the government is trying to obtain or achieve and allows contractor companies to respond via the RFI with information pertaining to how the contractor could satisfy objectives within the SOW. The action diagram facilitates question and answer sessions between contractor organizations, contracts team, and the project team. The final outcome is an RFI response from the contractor to the government contracting team.

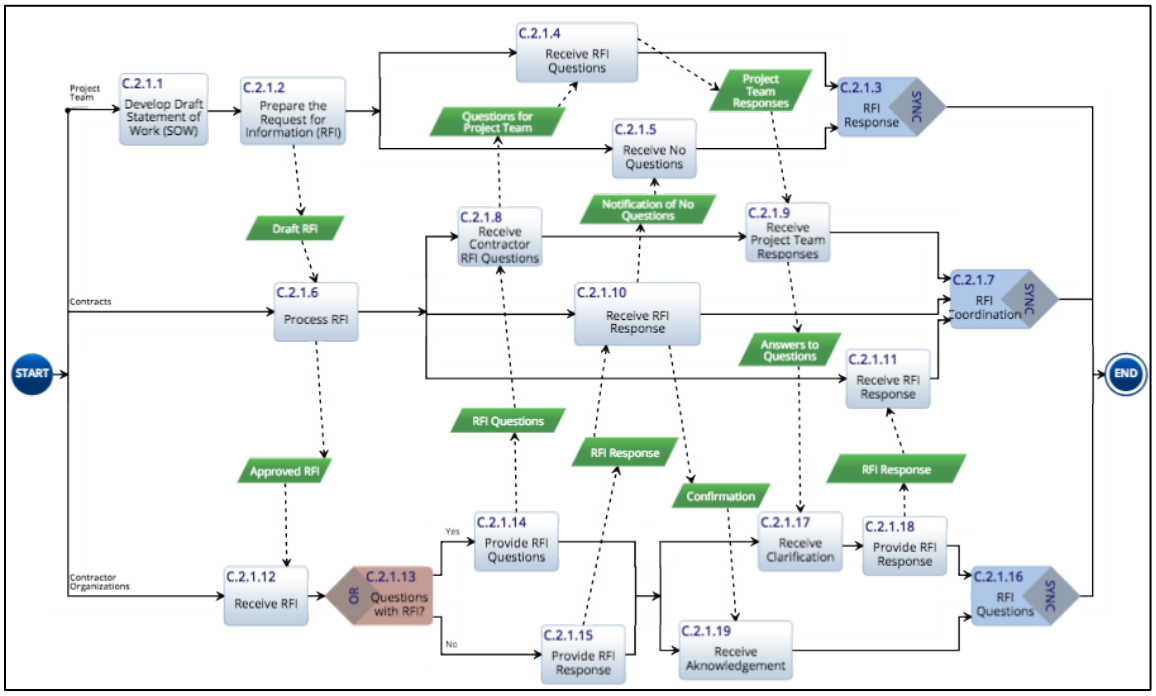



Figure 25. Decomposition of Perform Market Research (C.2.1)

## 2. Perform Preliminary Contracts Planning (C.2.2)

Based on the RFI responses received during market research, the project team will continue with preliminary planning and make the determination whether or not there is a need for an industry day. An industry day provides a venue for government and contracting organizations to meet and exchange relevant information about the technical needs of the

government as well as provide an opportunity for contractor organizations to ask questions about the potential work. The decomposition of “perform preliminary contracts planning (C.2.2)” was partitioned into three parallel paths, performed by the project team, contracts personnel, and contractor organizations (see Figure 26). If an industry day is required, the project team will make the request to the contracts team to contact industry partners and set up the industry day. Contractor organizations will receive the industry day notification and have the choice to participate or not. The project team will receive notification of the contractor’s intent to attend or not attend industry day. For those contractor organizations that attend industry day, there will be further technical and information exchanges between the project team and industry. If no industry day is requested by the project team, the contracts team will be informed and notify contractor organizations that no industry day will be scheduled before the release of the draft request for proposal (RFP).

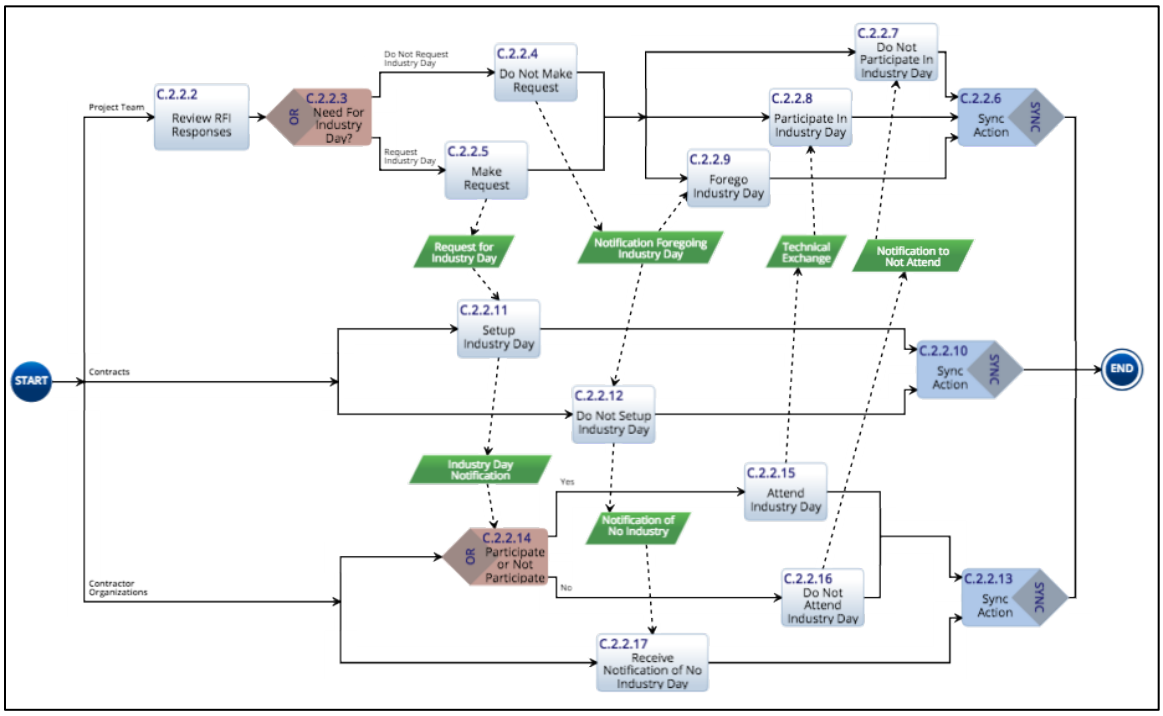



Figure 26. Decomposition of Perform Preliminary Contracts Planning (C.2.2)

### 3. Perform Draft RFP Activities (C.2.3)

After market research and industry day activities, the project team will finalize the technical components of the contracting task order package. This will include development of a performance work statement (PWS), contractor data requirements list (CDRLs), and an independent government cost estimate (IGCE). The decomposition of "perform draft RFP activities (C.2.3)" was partitioned into four parallel paths, performed by the project team, contracts personnel, contractor organizations, and template repository (see Figure 27). CDRLs are a list of requirements and instructions to the contractor for how, when, and what to deliver. The project team will utilize contracting templates and materials to help in this endeavor. "The template structure minimizes the overhead associated with the creation of multiple task orders" (Wrubel and Gross 2015, 31). Over time, the C2IS division captured best practices or lessons learned to help personnel write better contracts for software development. These lessons learned are incorporated into templates for the PWS and CDRLs to help the government ask for what is needed, receive all of what was paid for, verify the government gets what was paid for, and ensure it can reproduce what it paid for. This iterative process ends with the development of a draft RFP.

Personnel within contracts will receive the draft RFP and formally issue the draft RFP to contractor organizations. In parallel to the draft RFP development, the project team must also prepare and submit an information technology procurement request (ITPR). The ITPR will be submitted to an external agency, whose process is outside the scope of this thesis; however, it is still a valid action to capture. The ITPR is a mechanism for the broader U.S. Navy to track its information technology related procurements. Contractor organizations will receive the RFP, which typically begins an iterative question and answer process between contractors, the contracts team, and the project team (see Figure 27). The processes that personnel within contracts use to process and review the draft RFP are also outside the scope of this thesis.

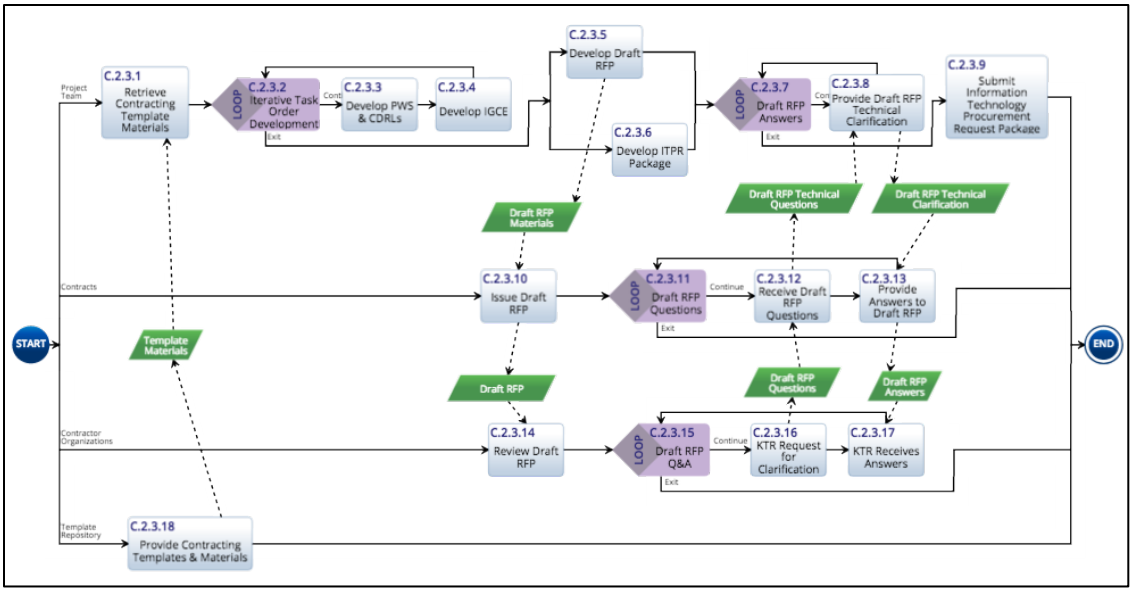



Figure 27. Decomposition of Perform Draft RFP Activities (C.2.3)

### 4. Perform Final RFP Solicitation and Award (C.2.4)

Once the draft RFP process is complete, the project team will update the RFP materials and submit the final RFP package to the contracts team. The decomposition of "perform final RFP solicitation and award (C.2.4)" was partitioned into three parallel paths, performed by the project team, contracts personnel, and contractor organizations (see Figure 28). In the same manner as the draft RFP, the contracts team will issue the final RFP and contractor organizations will receive, review, and begin the iterative question and answer process. After receiving any RFP clarification required, contractor organizations will submit a formal bid and proposal to the government contracts team. The contracts team will provide the contractor's technical proposal materials and the project team will perform a technical evaluation and submit that evaluation to the contracts team for review. The contracts team will perform a cost evaluation and once they receive the technical evaluation from the projects team, contracts will perform a tradeoff analysis. For example, one company may have been rated technically excellent, but their cost was significantly higher than a company rated as very good. The tradeoff analysis accounts for variance in technical rating and cost rating with the goal of obtaining the best value for the government. Once

the project team completes the technical evaluation, a purchase request must also be submitted in the organizations enterprise supply system (see Figure 28).

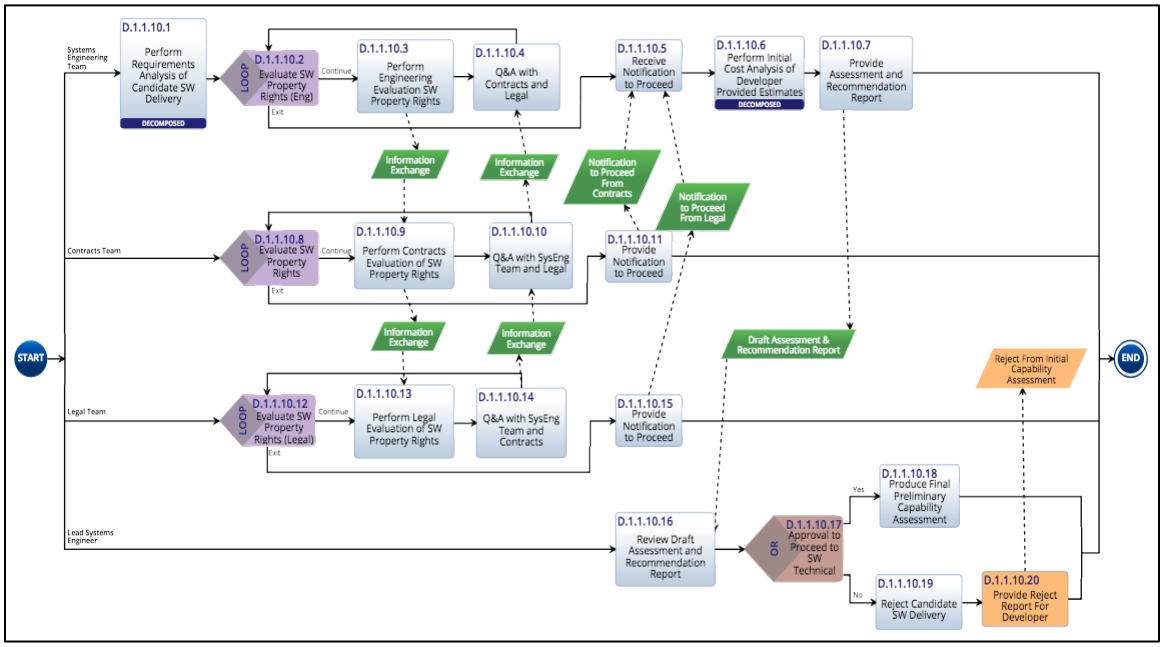



Figure 28. Decomposition of Perform Final RFP Solicitation and Award (C.2.4)

### 5. Award Task Order and Conduct COR Activities (C.2.5)

After the technical evaluation and tradeoff analysis is performed, the contract task order can be awarded. Prior to award, a contracting officer representative (COR) must be nominated and approved. This process is captured in the decomposition of "award task order and conduct COR activities (C.2.5)." This action was partitioned into three parallel paths, performed by the contracts personnel, the supervisor, and contracting officer representative (COR) (see Figure 29). A supervisor will nominate a COR and send the nomination to the contracting officer via the contracts team. Once approved, the COR will receive a formal designation letter from contracts. The task order is awarded, and the COR begins to perform his or her duties. The job of a COR is to track deliverables, monitor contractor performance, review contractor invoices, and provide performance feedback. COR activities will continuously occur throughout the life cycle of the task order.

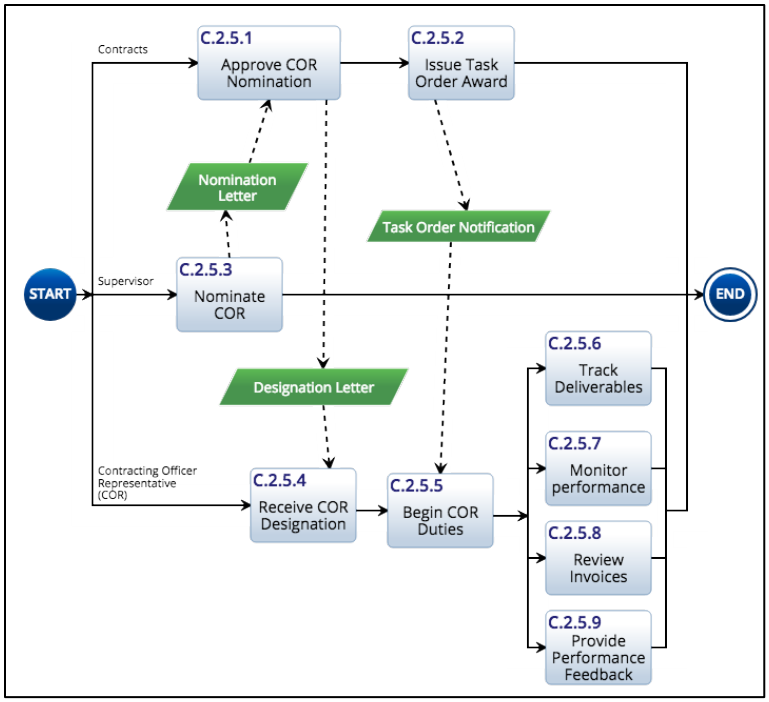



Figure 29. Decomposition of Award Task Order and Conduct COR Activities (C.2.5)

### 6. Develop and Administer Contract Task Orders for SW Development (EXT.F.2)

The actions from decomposition of "perform market research (C.2.1)," "perform preliminary contracts planning (C.2.2)," "perform draft RFP activities (C.2.3)," "perform final RFP solicitation and award (C.2.4)," and "award task order and conduct COR activities (C.2.5)," were used to develop the decomposed architecture diagram of "develop and administer contract task orders for software development (EXT.F.2)" (see Figure 30). As previously discussed, the purpose of contracting activities is to develop, compete, award, and monitor task orders that are used to supplement the government workforce.

There are five main functions and mechanisms within the decomposition (see Figure 30). The "template repository (EXT.C.2.1)" is the mechanism for "provide contracting template materials (EXT.F.2.1)." The function "perform market research and develop contracting materials (EXT.F.2.2)" is performed by the "project technical team (EXT.C.2.2)." The function "process and award contract task orders (EXT.F.2.3)" is performed by the "contract team (EXT.C.2.3)." The function "review SOW, RFP and

provide response (EXT.F.2.4)" is performed by "contractor organizations (EXT.C.2.4)." The function "provide post award admin and guidance (EXT.F.2.5)" is performed by the "contracting officer representative (COR) (EXT.C.2.5)."

The function "provide contracting template materials (EXT.F.2.1)" aligns to "perform draft RFP activities (C.2.3)." This function receives lessons learned as an input from "perform agile software development (F.0)" as well as suggestions for updated contracting templates from "perform market research and develop contracting materials (EXT.F.2.2)" and "process and award contract task orders (EXT.F.2.3)." Template materials are provided as an output for the project technical teams to use in performing market research and developing contracting materials.

The function "perform market research and develop contracting materials (EXT.F.2.2)" aligns to actions within "perform market research (C.2.1)," "perform preliminary contracts planning (C.2.2)," "perform draft RFP activities (C.2.3)," and "perform final RFP solicitation and award (C.2.4)." The function "process and award contract task orders (EXT.F.2.3)" aligns to actions within "perform draft RFP activities (C.2.3)," "perform final RFP solicitation and award (C.2.4)," and "award task order and conduct COR activities (C.2.5)." The function "review SOW, RFP and provide response (EXT.F.2.4)" aligns to actions within "perform market research (C.2.1)," "perform preliminary contracts planning (C.2.2)," "perform draft RFP activities (C.2.3)," and "perform final RFP solicitation and sward (C.2.4)." The function "provide post award admin and guidance (EXT.F.2.5)" aligns to conduct COR activities (C.2.5)."

The architecture reflects the iterative process modeled in the action diagrams for this function. Refined customer needs, software delivery information, the decision to outsource work, the draft RFI, industry day request, and draft RFP are all used to drive the final RFP package for contractor organizations to bid. Throughout this process, there are feedback loops included for questions and updating contracting template information. Once the contractor organizations submit their bid and proposal, the technical proposal materials will be provided from the contracts team to the project technical team for evaluation. The project technical team provides a technical evaluation of proposal output from "perform market research and develop contracting materials (EXT.F.2.2)" to the

contracts team via “process and award contract task orders (EXT.F.2.3).” After the contracts team reviews and adjudicates the task order award, notifications are provided to both contractor organizations and the contracting officer representative (COR). After the task order is awarded, the COR will provide post award administration and guidance to the contractor organizations. The COR also will act as the conduit between the project team and the performing contractors. The COR is also responsible for contractor invoice review and monthly status reports on contractor performance; therefore, software acceptance and rejection notifications are provided as an input to “provide post award admin and guidance (EXT.F.2.5).”

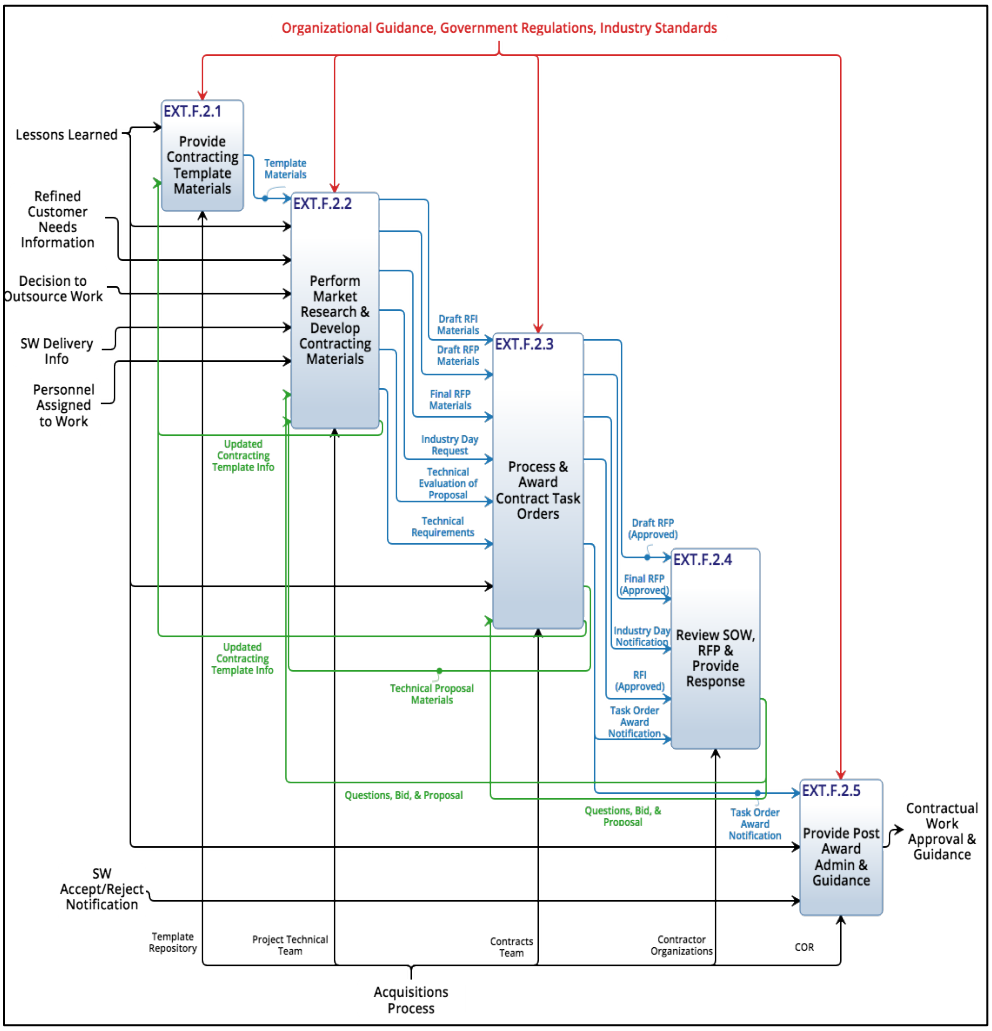



Figure 30. Architecture IDEF0 of Develop and Administer Contract Task Orders for SW Development (EXT.F.2)

## F. AGILE SOFTWARE DEVELOPMENT (D.1)

This section discusses the decomposition of the "perform agile software development (D.1)" action diagram and the "perform agile software development (F.0)" IDEF0. The action "agile software development (D.1)" decomposes into six actions: "perform capability assessment (D.1.1)," "establish templates and verify schedule (D.1.2)," "conduct software sprint (D.1.3)," "perform software quality engineering activities (D.1.4)," "perform continuous integration and testing (D.1.5)," and "conduct build release decision (D.1.6)" (see Figure 31). As previously discussed, agile software design and development focuses on software sprint activities and supporting activities that facilitate continuous software design, development, and testing.

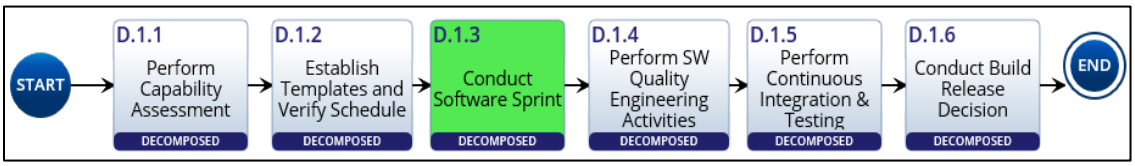



Figure 31. Decomposition of Perform Agile Software Development (D.1)

### 1. Perform Capability Assessment (D.1.1)

The decomposition of "perform capability assessment (D.1.1)" was partitioned into two parallel paths performed by the agile development, integration and test team, and government assessment team assets (see Figure 32). The action "perform capability assessment (D.1.1)" provides a gating process used for preliminary technical and cost assessments of an existing or science and technology (S&T) product or capability. If there is no existing capability or S&T product to being into the software development pipeline, then the agile development, integration and test team can proceed with new capability development. If an existing capability or S&T product is targeted for transition or integration into a broader system, then the capability assessment process is completed. The goal of performing the capability assessment process is to ensure the product or capability is mature enough to enter into the continuous development, integration, and test environment. Accepting immature S&T or existing products with major issues into the development pipeline can introduce technical risk and result in unexpected cost growth.

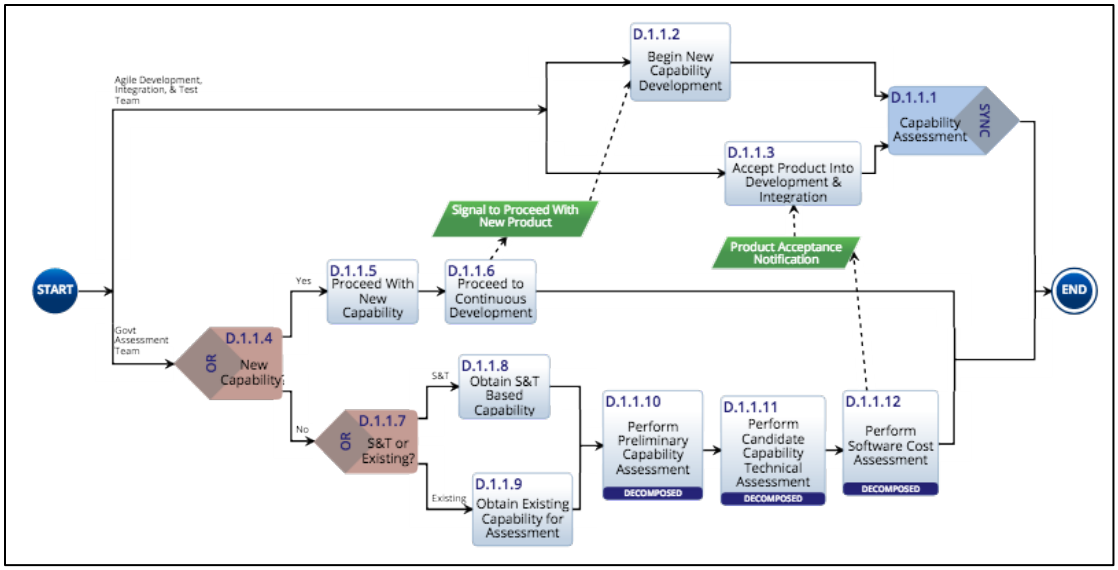



Figure 32. Decomposition of Perform Capability Assessment (D.1.1)

The decomposition of "perform preliminary capability assessment (D.1.1.10)" is partitioned into four parallel paths, which are performed by the system engineering team, contracts team, legal team, and lead systems engineer assets (see Figure 33). The preliminary capability assessment provides a precursory look into requirements analysis, property rights assessment, and an initial cost analysis of the developer provided estimates.

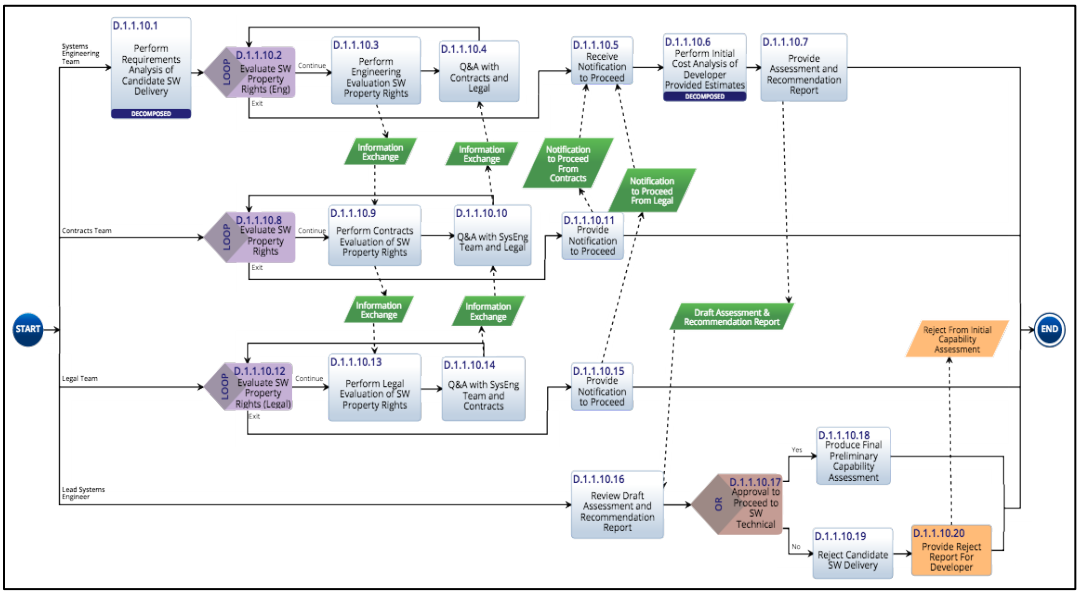



Figure 33. Decomposition of Perform Preliminary Capability Assessment (D.1.1.10)

The action “perform requirements analysis of candidate software delivery (D.1.1.10.1)” is performed by the system engineering team (see Figure 34). The purpose of this activity is to identify the product’s functional capabilities as stated by the developer or capability provider. These capabilities are then mapped to operational requirements and traced to system acquisition documents such as the initial capabilities document (ICD), capability development document (CDD) and software requirements specifications (SRS). The systems engineering team will identify any gaps or overlaps between system requirements and the product’s stated capabilities. An analysis of any external dependencies is performed to determine if there are potential risk factors for schedule, or cost, and technical performance such as unplanned integration with another system that the candidate product or capability dis not account for. The software quality engineering team will determine if the capability or candidate product will require modification of or additional cyber security or weapons accreditation and certification. Lastly, the system engineering team will provide a report documenting their findings based on analysis.

After the requirements analysis is performed, the systems engineering team will work with the contracts and legal team to perform an evaluation of software property rights (see Figure 33). This iterative process determines if the government has the desired level of software property rights to minimize future development and sustainment cost. The legal and contracts team will provide their recommendations to the systems engineering team for inclusion in the preliminary capability assessment report. Prior to the preliminary capability assessment report, the systems engineering team conducts an initial cost analysis (see Figure 35). The goal of “perform initial cost analysis of developer provided estimates (D.1.1.10.6)” is to conduct an initial analysis of the software licensing, development, post-development support, and software and system dependencies as stated by the capability developer. Based on the preliminary assessment of requirements, property rights, and cost, the government assessment team will make a determination of whether or not to proceed the next phase of a technical assessment of the candidate capability (see Figure 33). If the assessment team recommends rejecting the candidate capability, a reject report is produced, and a trigger is provided to the top-level “reject or redirect (R.1)” (see Figure 18). Unless

leadership or the sponsor mandate work progress, the process will then end with no further work performed.

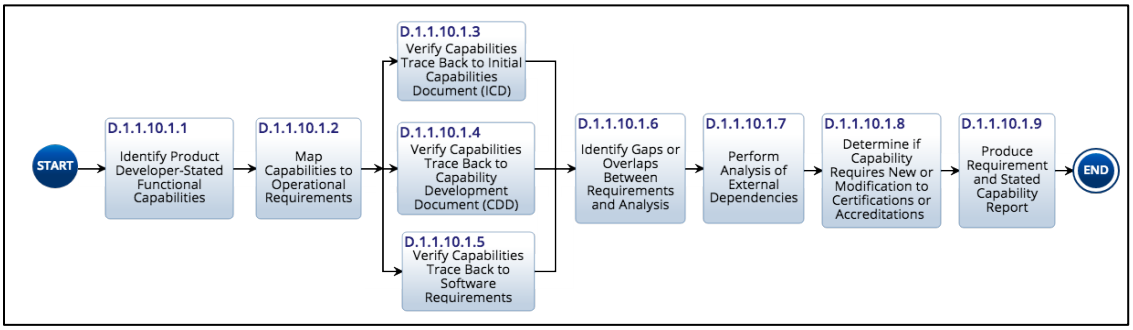



Figure 34. Decomposition of Perform Requirements Analysis of Candidate SW Delivery (D.1.1.10.1)

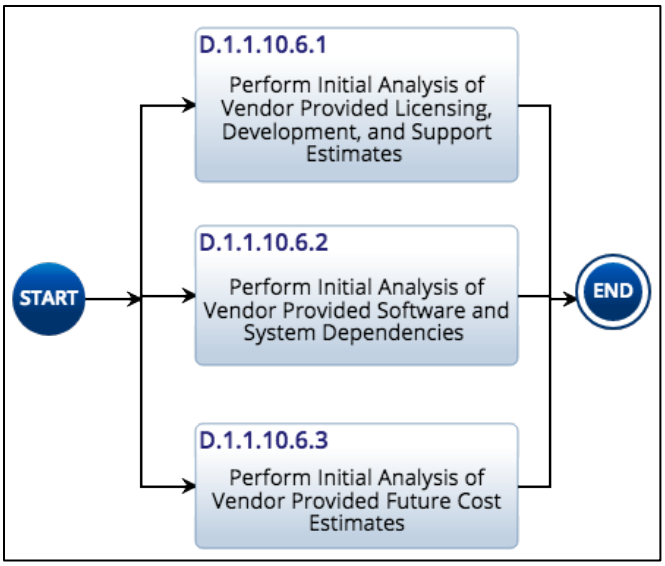



Figure 35. Decomposition of Perform Initial Cost Analysis of Developer Provided Estimates (D.1.1.10.6)

If the government assessment team makes a favorable recommendation after the preliminary capability assessment, the next action is to conduct a more detailed technical assessment of the candidate capability. The decomposition of "perform candidate capability technical assessment (D.1.1.11)" is partitioned into three parallel paths performed by cyber security subject matter experts (SMEs), software quality engineering

team, and lead systems engineer (see Figure 36). The cyber security SMEs will analyze the candidate capability or product delivery to verify proper security classification by the developer and run the software through vulnerability and cyber security analysis. If there are critical issues, the candidate delivery may be rejected without further work until major cyber issues are corrected. The cyber SME will produce an assessment report of all findings.

In conjunction with activities performed by the cyber security team, the software quality engineering team will proceed the action "complete in-depth technical assessment of candidate capability (D.1.1.11.4)" (see Figure 37). The software quality team will analyze the algorithms, data models, and software modeling languages used. The team will also analyze standards used as stated by the developer and perform a trial install and run of the candidate capability in a representative environment.

If the technical review results are favorable, cyber and technical assessments are produced and provided for use or reference in the next phase. If the technical assessment is not favorable, a recommendation to reject is produced and a trigger is provided to the top-level "reject or redirect (R.1)." As with the previous preliminary capability assessment actions, no further work is conducted unless leadership or the sponsor mandate work continue despite the technical issues.

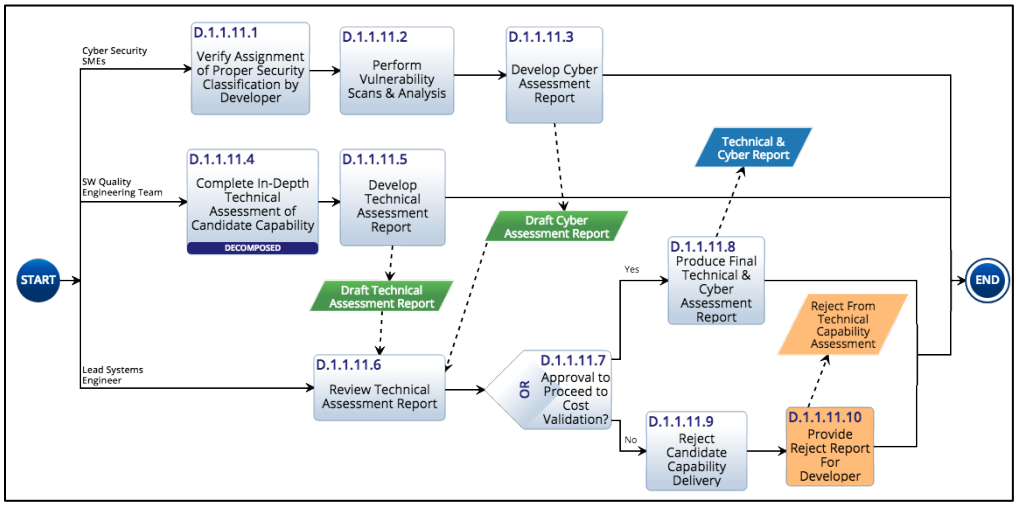



Figure 36. Decomposition of Perform Candidate Capability Technical Assessment (D.1.1.11)

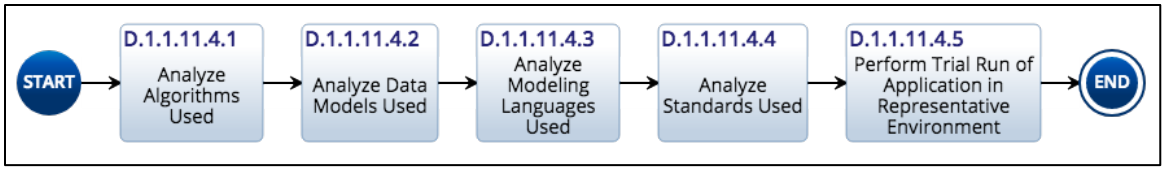



Figure 37. Decomposition of Complete In-Depth Technical Assessment of Candidate Capability (D.1.1.11.4)

If the preliminary capability and technical assessments are favorable, the government assessment team will perform a more in-depth cost assessment. The decomposition of "perform software cost assessment (D.1.1.12)" is partitioned into two parallel paths, performed by the systems engineering team and lead systems engineer (see Figure 38). The systems engineering team will review the cost estimate provided by the capability developer and in parallel, perform an independent government cost estimate. If the developer is a government only team, and no contractors are or were used, the government still performs an independent estimate and compares it to the government developer cost estimate. Ideally, the two estimates will be similar. If the developer's cost estimate is significantly higher or lower than the government's independent estimate, further analysis is required. For both acceptable and unacceptable cost structure, a cost validation or rejection report is generated and provided to the lead systems engineer for review and determination of whether to proceed with further software development, integration, and test. As with previous assessments, leadership or a sponsor may determine that the risk or issues with cost structure is acceptable and direct further development activities to proceed. If a rejection is substantiated, a reject trigger is provided to the top-level "reject or redirect (R.1)" and the process ends.

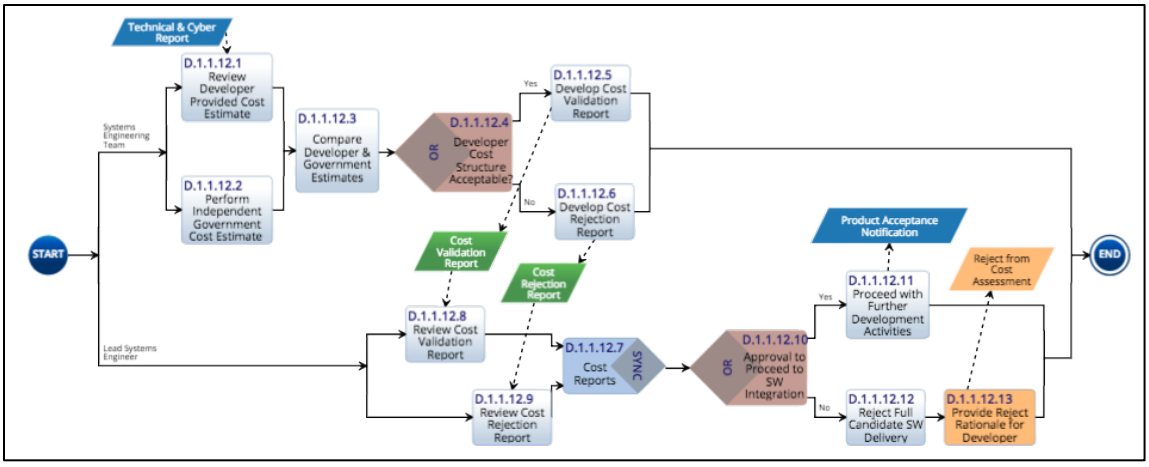



Figure 38. Decomposition of Perform SW Cost Assessment (D.1.1.12)

### 2. Establish Templates and Verify Schedule (D.1.2)

Once the capability or product is formally accepted, the configuration management and development teams will work together to establish and verify templates and schedules for product deliveries that will take place during software sprints. While these templates are part of CDRLs discussed in Chapter III Section E for Contracts Development, the CDRLs specify what, when, and how, but may not descriptive enough to ensure alignment of expectations between the government and contractor developer. Establishing templates and verifying schedules helps to mitigate this potential issue. The decomposition of "establish templates and verify schedule (D.1.2)" was partitioned into two parallel paths, performed by the configuration management (CM) team and agile development, integration and test team (see Figure 39). This is an iterative process between the development team and the CM team until consensus is reached by the CM and development teams. The CM team ultimately maintains configuration of all documents, deliverables, and templates.

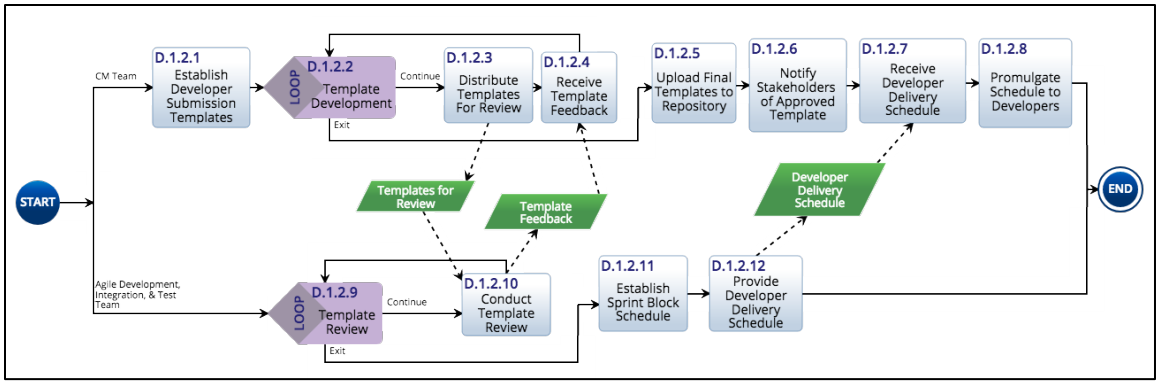



Figure 39. Decomposition of Establish Templates and Verify Schedule (D.1.2)

### 3. Conduct Software Sprint (D.1.3)

After the completion of "perform capability assessment (D.1.1)" and "decomposition of establish templates and verify schedule (D.1.2)," the next actions encompass activities for agile software development via backlog and sprint planning, sprint execution, and sprint review. The actions decomposed within "conduct software sprint (D.1.3)" are performed by multiple assets. The purpose of these assets and actions are explained in each decomposition of "initial backlog and sprint planning (D.1.3.1)," "perform spring planning, execution, and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)" (see Figure 40).

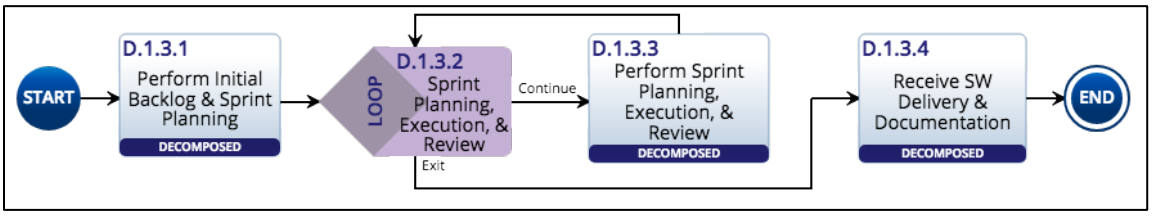



Figure 40. Decomposition of Conduct Software Sprint (D.1.3)

One of the first actions to support software sprints includes the initial planning for sprint activities and initialization of the product backlog, which is a continuously updated list of software items that require further development. If a task cannot be completed during a specified sprint, it will be added or maintained in the product backlog for future sprint

consideration. Within the agile construct, planning, development, and testing activities are continuous; however, for the purposes of this model, the upfront planning activities were broken out from the continuous sprint planning, execution, and review activities. The decomposition of "perform initial backlog and sprint planning (D.1.3.1)" is performed by the sprint planning team (see Figure 41). The type and quantity of team member varies based on the action to be performed. The action "define roles and responsibilities (D.1.3.1.1)" will establish and communicate each team member's part in the sprint activities and what each team member is responsible for. The action "conduct build review and release planning (D.1.3.1.2)" focuses on the software epics and mapping features to those epics in order to establish a software roadmap plan for the product or capability to be developed. Epics contain large amounts of work that cannot be completed in a single sprint. Epics are further decomposed into smaller sets of requirements known as user stories. The scope of "initialize product backlog (D.1.3.1.3)" will vary based on the size of the development effort. For the project used to support this thesis, there were 530 system requirements. A higher or lower quantity and complexity of system requirements may drive larger or smaller team size for initialing the product backlog. Although initializing the product backlog is an effort and resource intensive activity, it is typically done once for a product baseline. Subsequent actions will update the product backlog throughout sprint execution and review.

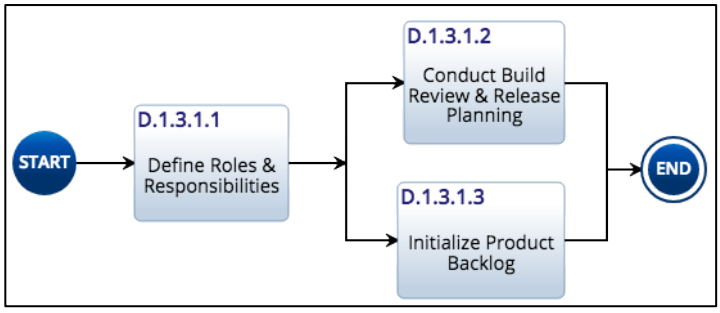



Figure 41. Decomposition of Initial Backlog and Sprint Planning (D.1.3.1)

After the roles and responsibilities, build review, release planning, and initial product backlog are established, the continuous sprint planning, execution, and review cycle will begin. The decomposition of “perform sprint planning, execution, and review (D.1.3.3)” was partitioned into three parallel paths and is performed by the sprint planning team, sprint execution team, and sprint review team (see Figure 42). This activity also occurs in a loop (see Figure 40). The first action allows for the prioritization of the product backlog. The goal of prioritization is to review the software roadmap against sprint tasks and determine the priority of epics and user stories to be worked. The product backlog is also updated in parallel based upon feedback during sprint execution, review, and retrospective. The product backlog can be updated to account for new information or technologies that have occurred since the previous sprint. Once the backlog is prioritized and updated, the sprint planning team conduct planning to allocates user stories to the sprint.

After sprint planning, the sprint execution team is notified to begin development, shown in the model as “conduct sprint (D.1.3.3.5)” (see Figure 43). Details and model flow for sprint execution are discussed immediately following “decomposition of perform sprint planning, execution, and review (D.1.3.3).” The sprint review team will conduct a sprint review and retrospective after the development cycle is complete (see Figure 42). During the sprint review, the development team will present the latest product version to stakeholders and obtain feedback. The retrospective is more of an internal project team review to capture lessons learned. Following the sprint retrospective, the project team may conduct what is known as a Scrum of Scrums. This action is used when there are multiple Scrum teams working parallel development efforts and is needed to have shared awareness across the team. As previously stated, feedback from these actions is used to update and prioritize the product backlog for the next sprint; however, there can be a lag of one or two sprints by the time sprint feedback is incorporated into planning for a follow-on sprint. This is because as sprint feedback is collected, reviewed, and built into the backlog, the next sprint will have already begun.

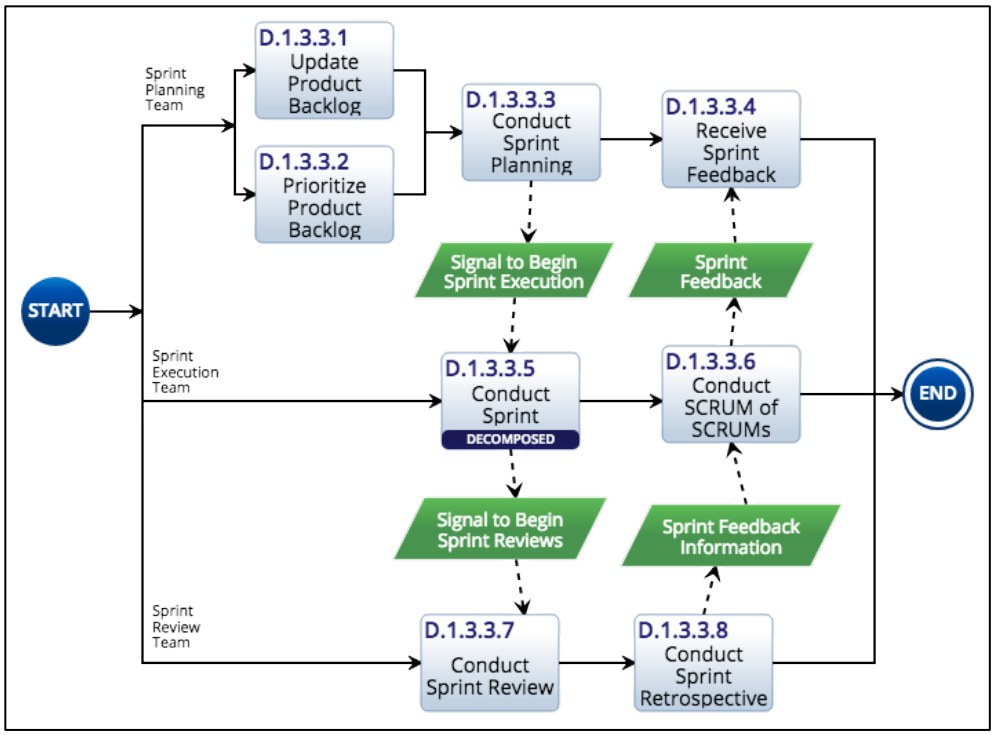



Figure 42. Decomposition of Perform Sprint Planning, Execution, and Review (D.1.3.3)

Once the sprint planning is complete, the software developers will conduct the sprint. The decomposition of "conduct sprint (D.1.3.3.5)" was partitioned into three parallel paths and is performed by the sprint planning team, the software developer, and project manager (see Figure 43). The development cycle within the project analyzed for this thesis is planned for either 10 or 15 days. Daily Scrums are held to discuss progress and roadblocks within the planned sprint activities. During the Scrum, the team discusses and determines an appropriate course of action if there are issues. Typically, a decision is made to either increase, decrease, or modify the scope of the some or all of the sprint. Conversely, the product backlog can be analyzed again and the scope of the sprint can be modified to swap out a user story with another one that is equal or less effort. In addition, the scope may also be increased. An increase in scope may require adding user stories to the sprint, adding resources such as additional developers, or increasing the number of user stories and resources.

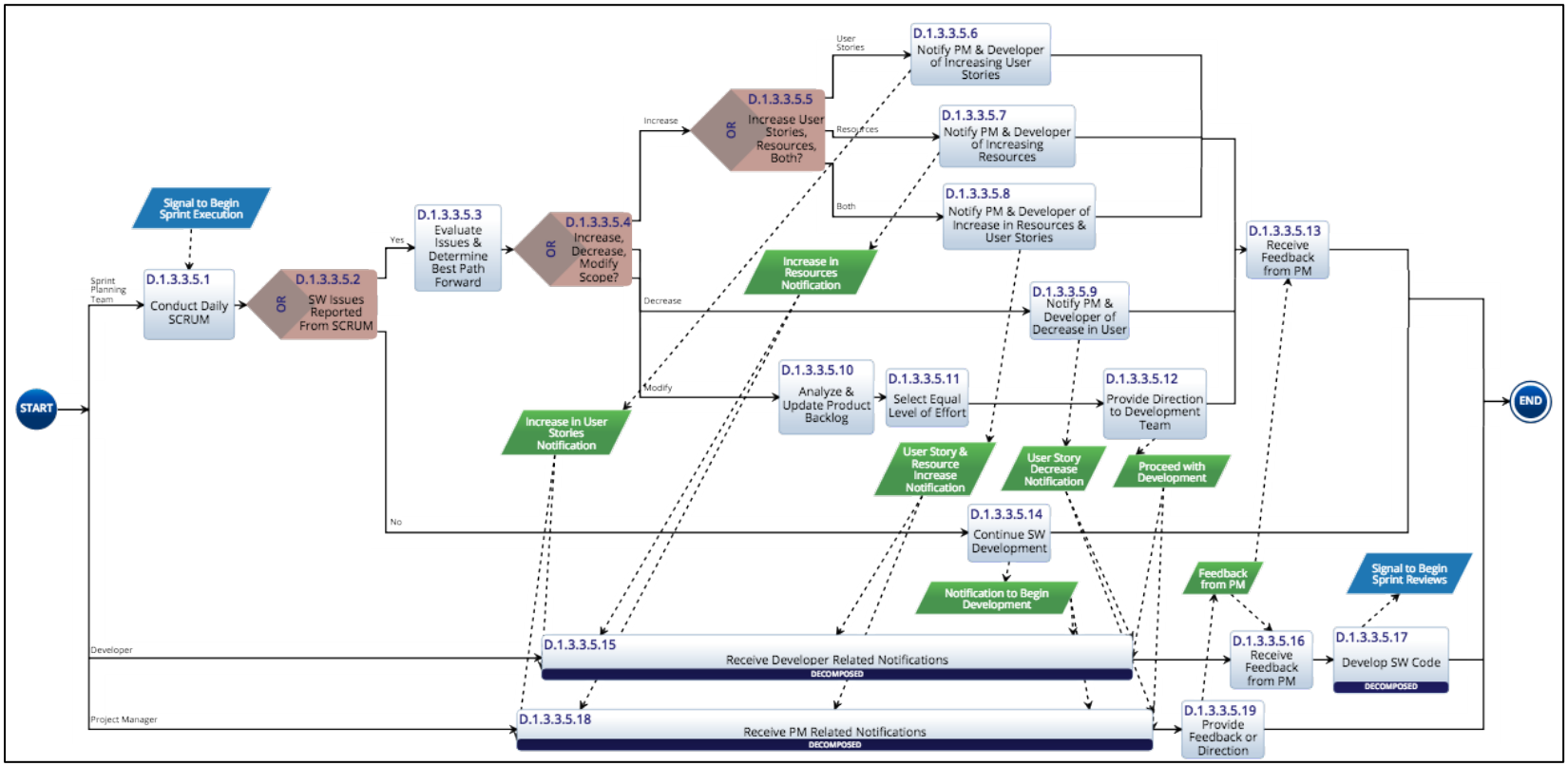



Figure 43. Decomposition of Conduct Sprint (D.1.3.3.5)

The sprint planning team notifies the PM and developers of the viable path forward. The decomposition of “receive developer related notifications (D.1.3.3.5.14)” (see Figure 44) and the decomposition of “receive PM related notifications (D.1.3.3.5.16)” (see Figure 45) describe how the developer and project manager receive notifications. The project manager can influence the course of action taken. If the PM needs to adjust what the sprint planning team is recommending, he or she will notify the sprint planning team, who will in turn make adjustments to the spring plan and notify the sprint execution team (see Figure 43). The decomposition of “receive developer related notifications (D.1.3.3.5.14)” was partitioned into five parallel paths and all paths are performed by the software developer (see Figure 44). Only one action from all available paths will be performed. The model was constructed in this manner for model styling purposes to synchronize with its parent diagram “decomposition of conduct sprint (D.1.3.3.5)” (see Figure 43).

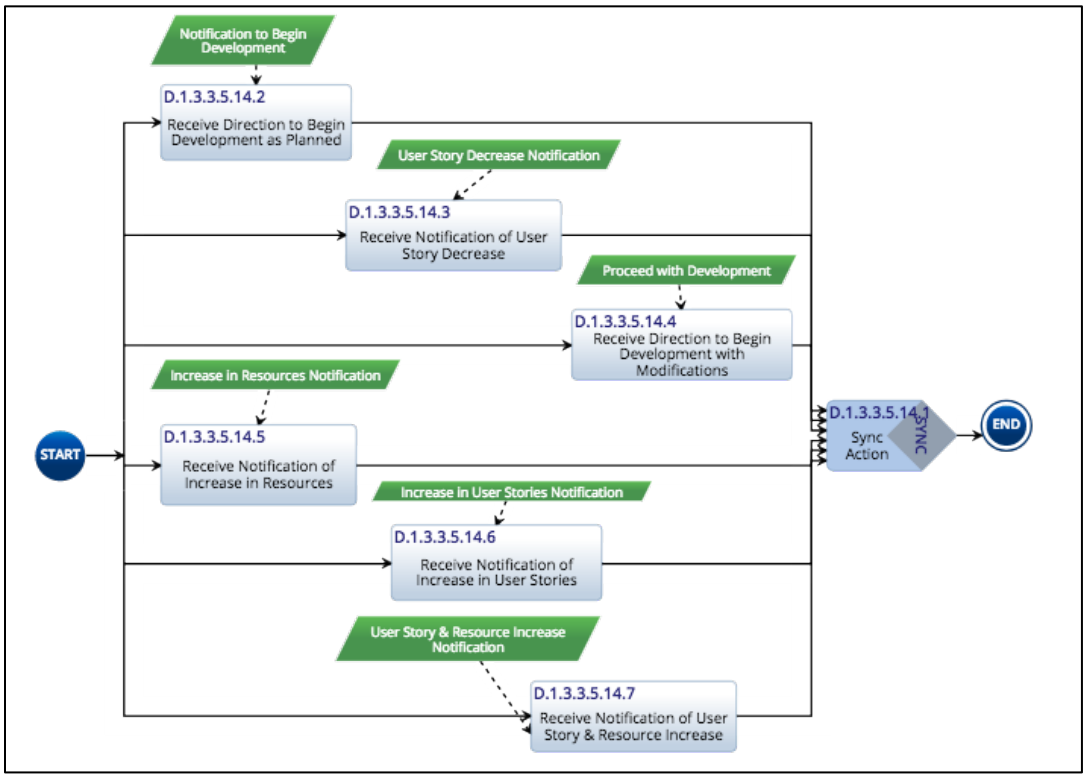



Figure 44. Decomposition of Receive Developer Related Notifications (D.1.3.3.5.14)

The decomposition of “receive PM related notifications (D.1.3.3.5.16)” was partitioned into five parallel paths and all paths are performed by the project manager (see Figure 45). Only one action from all available paths will be performed. As with the previous model, the model was constructed this way for model styling purposes to synchronize with and streamline its parent diagram “decomposition of conduct sprint (D.1.3.3.5)” (see Figure 43).

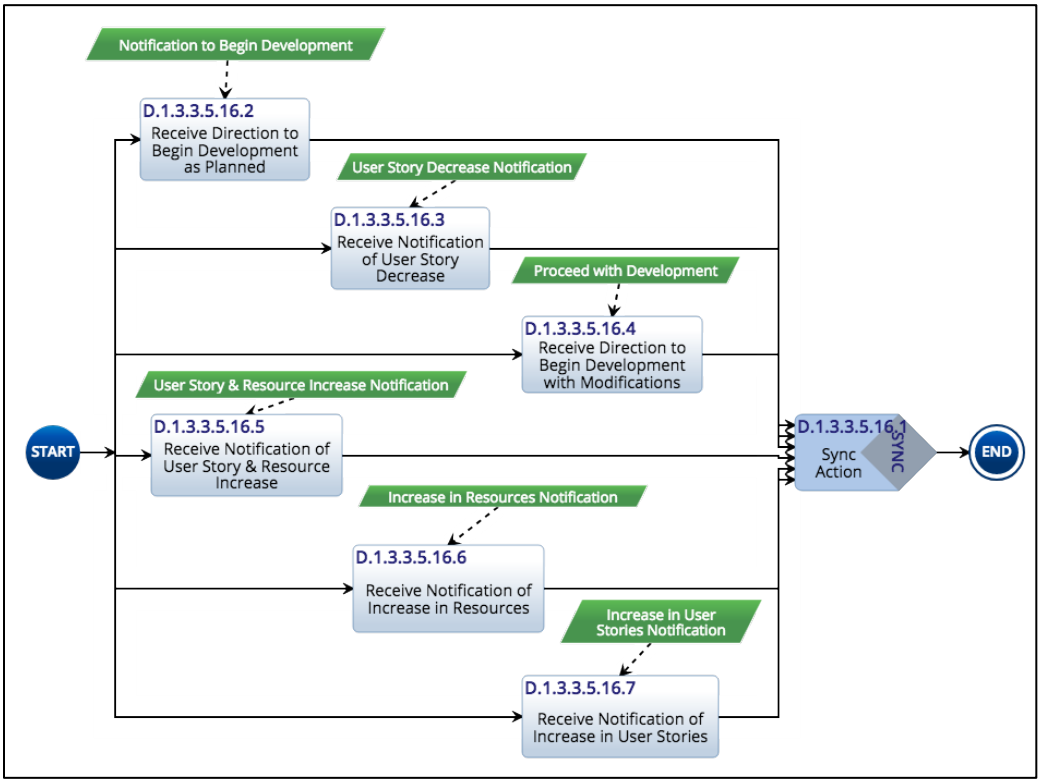



Figure 45. Decomposition of Receive PM Related Notifications (D.1.3.3.5.16)

The decomposition of “develop software code (D.1.3.3.5.15)” is performed by the software development team (see Figure 46). Within the C2IS division’s project for this thesis research, agile software sprints are time-bound to either 10- or 15-day sprint cycles. The loop actions of developing software code, completing peer review of software, and committing code to a centralized repository will continuously occur over the course of

either 10 or 15 days. At the conclusion of the sprint, the sprint review team will be notified to being sprint review activities such as a Scrum of Scrums and sprint retrospective.

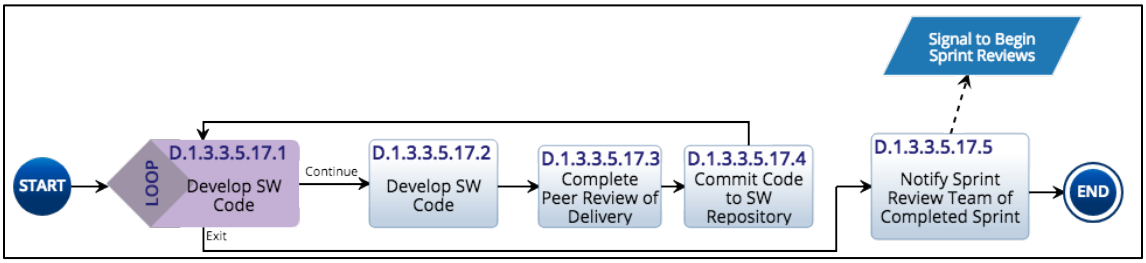



Figure 46. Decomposition of Develop SW Code (D.1.3.3.5.15)

The software will go through a formal delivery acceptance review upon completion of the sprint. The decomposition of "receive software delivery and documentation (D.1.3.4)" was partitioned into three parallel paths and is performed by the configuration management team, software developer, and lead systems engineer (see Figure 47). "Review delivery against acceptance criteria (D.1.3.4.1)" starts the acceptance process with a review of all deliverables (see Figure 48). The acceptance criteria for a software delivery verifies: delivery of license agreements; inclusion of intellectual property rights information; build instructions for the software; virus scans; verification of source code; applicable scripts required for future integration; requirements documentation; software design documentation; interface requirements; and test plans with reports. If there are no issues with the acceptance criteria, the CM team will accept the delivery in full and software quality engineering activities will proceed. The CM team will notify the developer and lead systems engineer if there are issues with the delivery. The lead systems engineer will review the issues and determine whether or not they are minor enough to grant a waiver and proceed with delivery acceptance. If the lead engineer grants a waiver, it is annotated with the delivery and CM accepts the delivery with an approved waiver, notifying the developer of the waiver and any corrective action. If the issues are significant enough to avoid accepting the delivery, the lead engineer notifies the CM team and developer that the delivery will be rejected. If rejection is required, a reject trigger is provided to the top-level "reject or redirect (R.1)" and the process ends (see Figure 47).

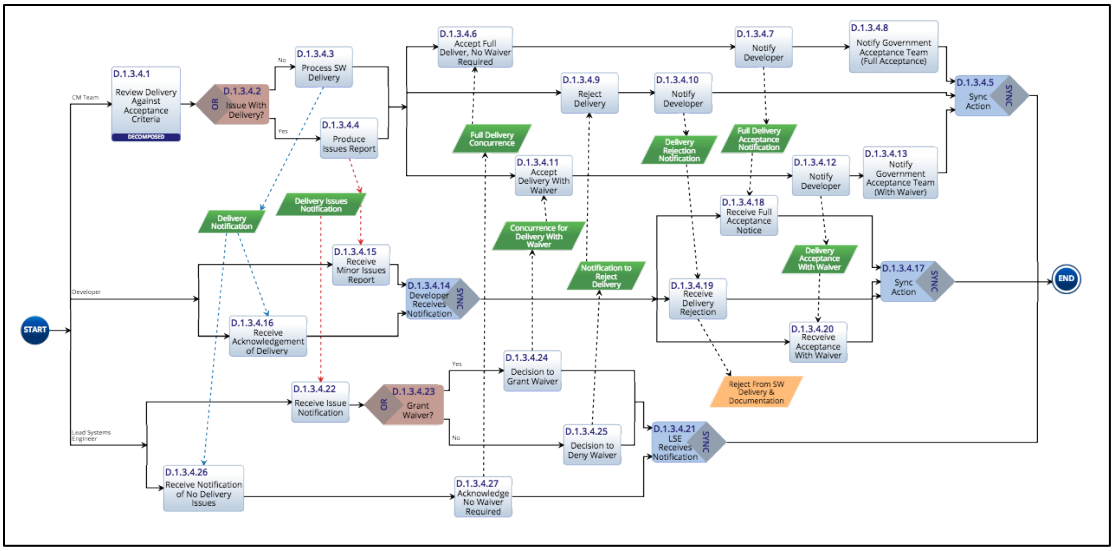



Figure 47. Decomposition of Receive SW Delivery and Documentation (D.1.3.4)

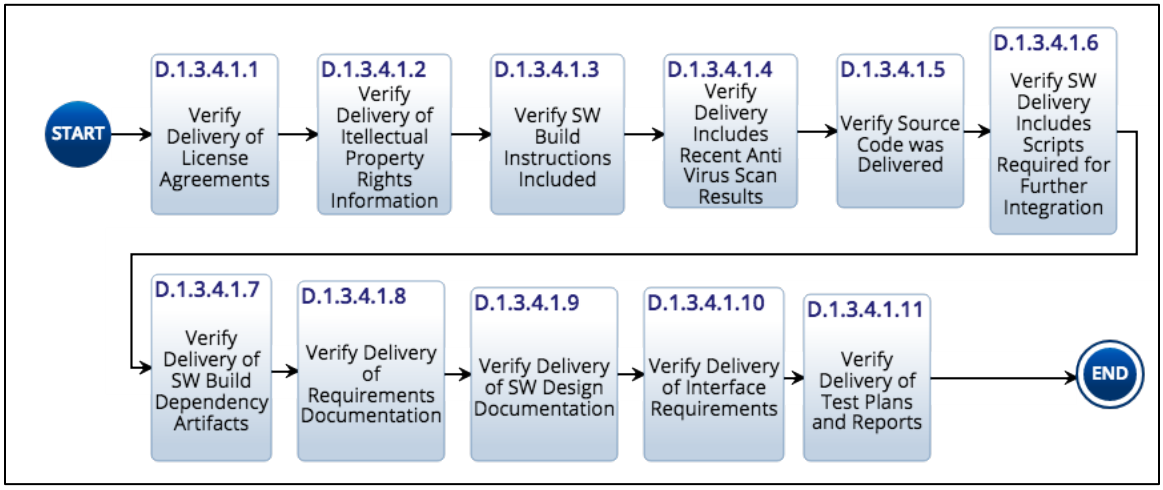



Figure 48. Decomposition of Review Delivery against Acceptance Criteria (D.1.3.4.1)

### 4. Perform SW Quality Engineering Activities (D.1.4)

Once the software is accepted via the delivery process, the next set of actions is used to determine if there are any quality issues with the software. “Perform software quality engineering activities (D.1.4)” is decomposed into three actions: “perform software cyber vulnerability scan (D.1.4.1),” “conduct software quality analysis (D.1.4.2),” and “build executables from source code (D.1.4.3)” (see Figure 49). The details of each action

are discussed within its decomposition and the software quality engineering team performs or leads all actions within “perform software quality engineering activities (D.1.4).”


Figure 49. Decomposition of Perform SW Quality Engineering Activities (D.1.4)

The action “perform software cyber vulnerability scan (D.1.4.1)” begins with verifying security classification markings are compliant. Next, the software is scanned for cyber vulnerabilities (see Figure 50). The team will determine if a full scan or partial scan is required depending on whether or not the delivery is new, or if only a portion of the code was modified. If the software quality engineering team finds no vulnerabilities, they will proceed with the next action for conducting the software quality analysis. If vulnerabilities are discovered, the team has to consider whether work can progress with the cyber vulnerabilities found. If work can progress, the software quality engineering team can retrieve the source code from the software repository, perform corrective actions to the code, and re-commit the code to the software repository. The severity and complexity of the changes required will determine whether this action is taken or not. In this context, “cyclomatic complexity provides a quantitative measure of the logical complexity of a program. It is the upper bound for the number of tests that must be conducted to assure that all statements have been executed at least once” (Osmundson and Giammarco 2017). Another possibility is that the vulnerabilities are so severe that work cannot proceed. If the team makes this determination, a reject trigger is provided to the top-level “reject or redirect (R.1)” and the process ends (see Figure 50). An additional possibility other than rejecting the code or fixing it immediately, is to proceed with subsequent actions and work through the deficiency reporting and engineering change request processes. The deficiency reporting process is discussed in Chapter III, Section F Subsection 5. This option may be

warranted when the software still contains vulnerabilities yet the updated software to be fielded fixes critical vulnerabilities elsewhere.

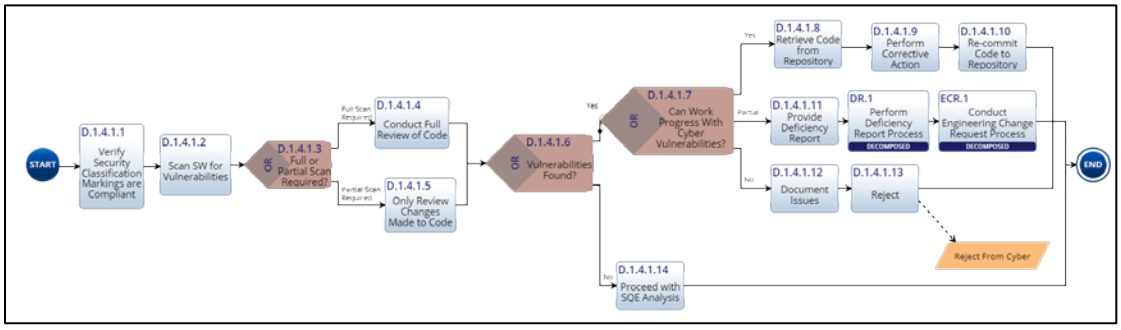



Figure 50. Decomposition of Perform SW Cyber Vulnerability Scan (D.1.4.1)

Upon completion of the software cyber vulnerability actions, the software will undergo an in-depth quality analysis. As with the cyber actions, “conduct software quality analysis (D.1.4.2)” is performed by the software quality engineering team (see Figure 51). The deficiency report and engineering change request processes initially discussed within the cyber vulnerability analysis are also applicable in the software quality analysis. Since these actions are built to be modular and can be inserted as needed to facilitate a deficiency report or engineering change request, they are not repeated in discussion for the software quality analysis actions.

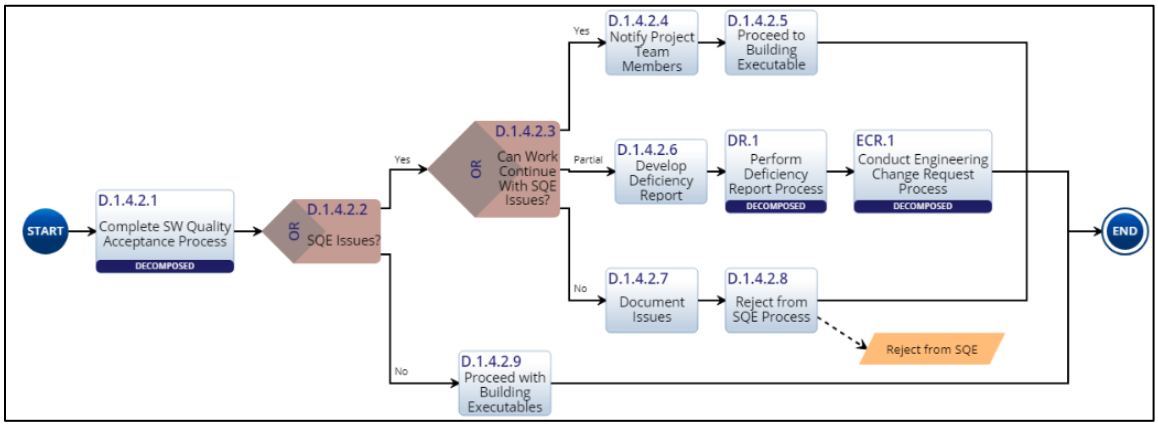



Figure 51. Decomposition of Conduct SW Quality Analysis (D.1.4.2)

The main activity for software quality analysis is performing quality acceptance checks on the software code. The decomposition of "complete software quality acceptance process (D.1.4.2.1)" illustrates 15 unique quality checks performed by the software quality engineering team (see Figure 52). The purpose of these quality checks is to ensure a quality product or capability is delivered to the end-user, minimize risk of costly post-production fix and repair cycles, and reduce sustainment cost in the product's life cycle.

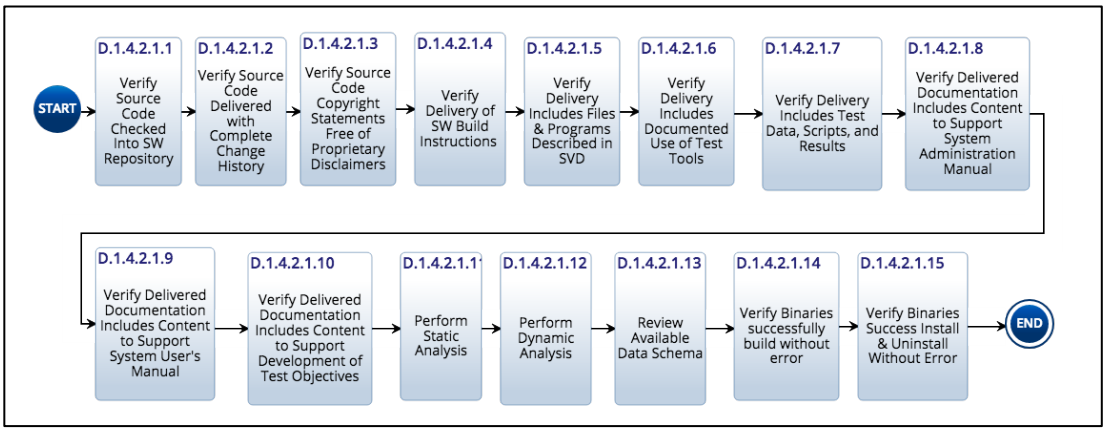



Figure 52. Decomposition of Complete SW Quality Acceptance Process (D.1.4.2.1)

Once the software quality analysis is performed, the next action is to ensure executable software can be built from the delivered source code. The software quality engineering team also performs "build executables from source code (D.1.4.3)" (see Figure 53). If there are no issues building executables, the software will be installed on the integration and test suite of equipment for further analysis. If there are issues building executables, the team may perform corrective action is the issue is minor enough, then re-commit the code to the repository. If work cannot proceed due to significant issues building the executables, then the issues are documented and the software is rejected via a trigger back to the top-level "reject or redirect (R.1)" ending the work.

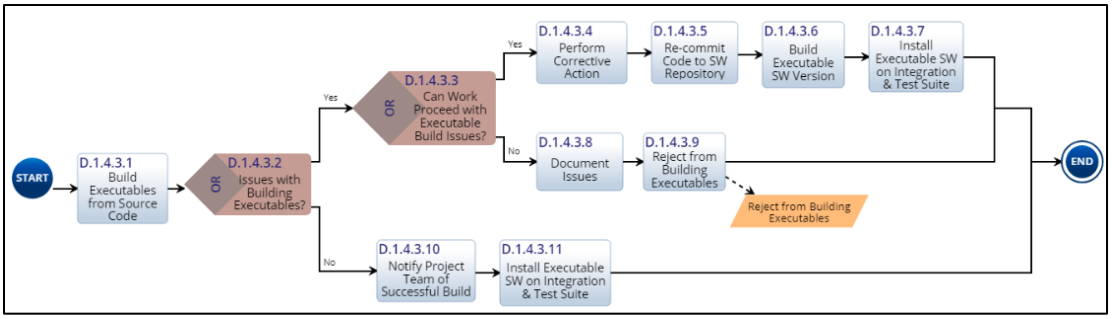



Figure 53. Decomposition of Build Executables from Source Code (D.1.4.3)

## 5. Deficiency Report Process (DR.1)

If the software quality engineering team decides to submit a deficiency report and work through the engineering change request process, the deficiency will be documented and evaluated at the next engineering review board (ERB) to determine if the deficiency is valid. The action “perform deficiency report process (DR.1)” models the deficiency report process (see Figure 54). If ERB determines that the deficiency is invalid, it will be closed in the reporting system. If the ERB determines that the deficiency is valid, the ERB will also investigate whether or not the deficiency is within the software developer’s scope to address. If the deficiency is within scope, the ERB will work with the developers to determine is the deficiency is already assigned to a future sprint. If it is already assigned to a future sprint, the deficiency report will be monitored until the issue is corrected. If it is not already assigned to a future sprint, the deficiency will be used to update the product backlog for incorporation into a future sprint. If it is determined that the deficiency is outside the scope of the developer, ERB will determine the appropriate action (see Figure 54 and Figure 55).

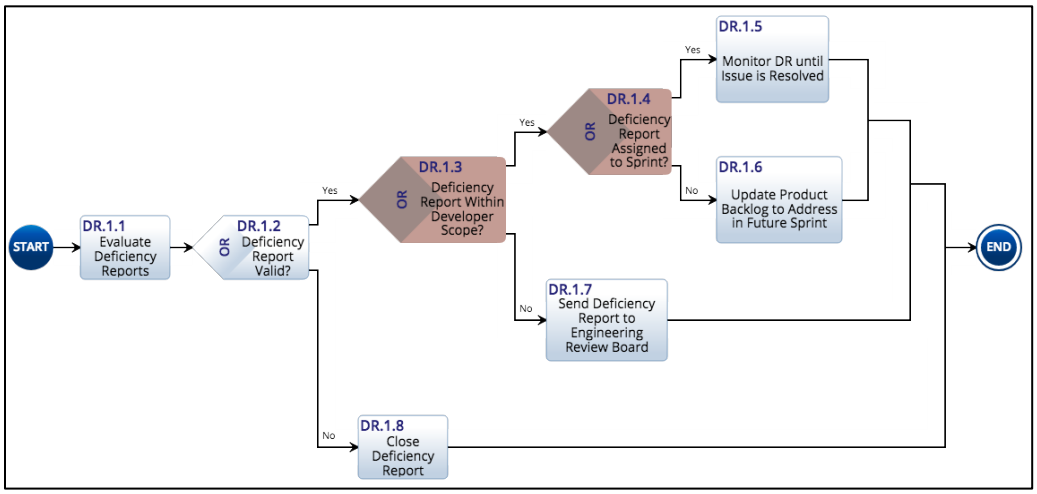



Figure 54. Decomposition of Perform Deficiency Report Process (DR.1)

## 6. Engineering Change Request (ECR.1)

The "conduct engineering change request process (ECR.1)" is performed by the engineering review board (ERB) (see Figure 55), which is initiated by the previously described deficiency report process. The ERB enables review and adjudication of DRs that are elevated by the product owner and cannot be resolved during an active sprint. The decomposition of "conduct engineering change request process (ECR.1)" yields three sets of activities that occur in a progressive series, and they include "prepare for ERB (ECR.1.1)," "conduct ERB (ECR.1.2)," and "close ERB (ECR1.3)." The ERB Chair, configuration management team, and ERB team perform their corresponding actions within each decomposed activity. Within ECR.1, there are three primary activities (see Figure 55).


Figure 55. Decomposition of Conduct Engineering Change Request Process (ECR.1)

The action “prepare for ERB (ECR.1.1)” is the first step in the ERB process, and it includes the activities associated with preparing for the ERB meeting. The decomposition of “prepare for ERB (ECR.1.1)” is partitioned into three parallel paths that are respectively performed by the ERB Chair, configuration management team, and ERB Team (see Figure 56). The ERB preparation activities begin when the ERB Chair reviews DRs for completeness and clarity and recommends items for the CM team to add to the ERB agenda. Once the CM team receives the ERB Chair’s proposed items for the ERB agenda, they draft an ERB agenda, which includes the date, location, connectivity information, discussion items, and DRs for adjudication. After the draft is complete, the CM team provides it to the ERB Chair for their review and approval for distribution approval. The ERB Chair then provides the finalized agenda for distribution to the CM team. The CM team then distributes the finalized agenda and supporting materials to corresponding ERB team members. Once the ERB team receives the agenda and supporting materials they conduct review in preparation for the ERB meeting.

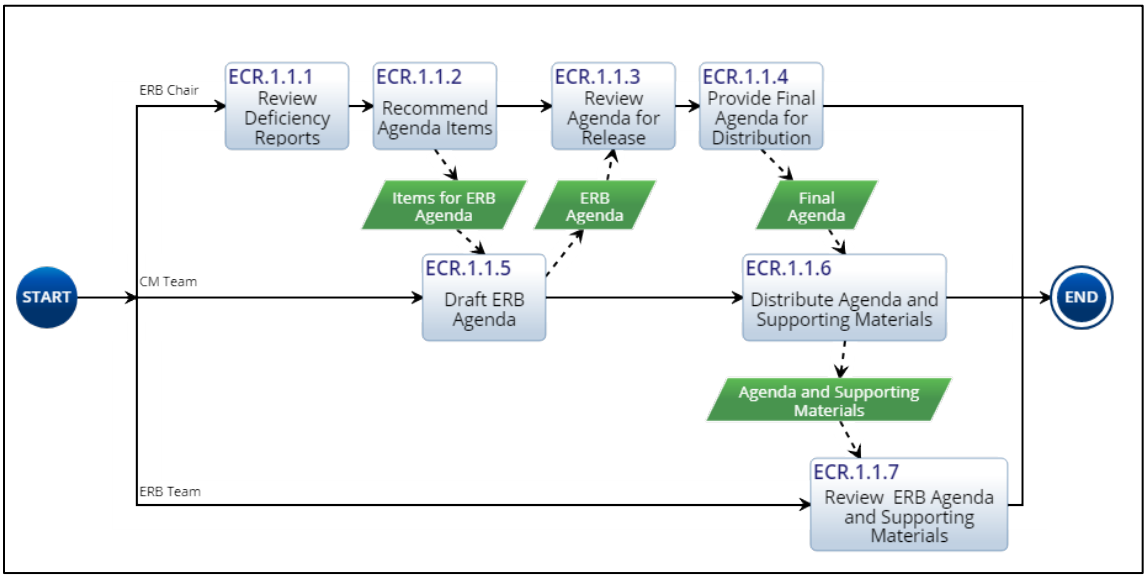



Figure 56. Decomposition of Prepare for ERB (ECR.1.1)

The activities associated with “conduct ERB (ECR.1.2)” start as scheduled (see Figure 57). The first step during the ERB meeting is for the CM team to take attendance

and confirm that a quorum consisting of essential participants is present. If the ERB chair, the ERB chair's designated replacement, or quorum is not present, then the CM team reschedules the ERB. If a quorum is present, then the ERB meeting can proceed as planned, and the CM Team notifies the ERB Chair to start the meeting. Subsequently, the ERB chair leads review of the agenda items, during which they can defer lower priority items to ensure the meeting stays within time constraints. Additionally, review of a DR may be deferred from ERB if the party responsible for submitting and presenting the DR is not present. During the ERB meeting, the ERB Team is responsible for providing adjudication for each presented DR. Adjudication involves advising approval, disapproval, deferral, or approval with action items based on the ERB team's assessment of the respective presented DR. Following adjudication, the ERB Chair evaluates the meeting details to assign associated actions with priorities and suspense dates where applicable. The CM team takes note of the advised DR adjudication, action items, and topics from the meeting. The proposed DR adjudication will be executed in "close ERB (ECR 1.3)."

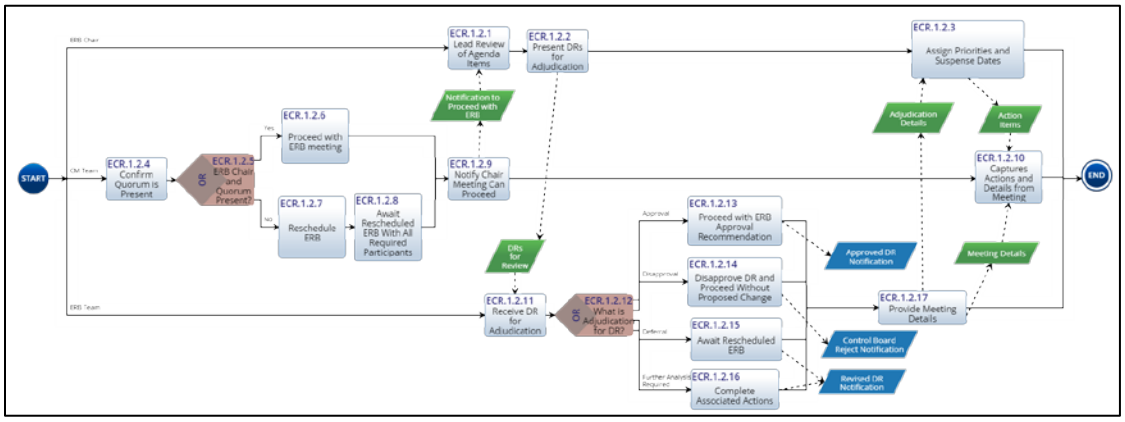



Figure 57. Decomposition of Conduct ERB (ECR.1.2)

The ERB Chair, configuration management team, and ERB team complete required steps to close out the ERB in three parallel paths in "close ERB (ECR 1.3)" (see Figure 58). In the initial step to close out ERB, the CM team processes each DR based on its adjudication. If the CM team receives notification that the DR needs to be revised, then the DR is scheduled for a subsequent ERB. If the CM team receives notification the DR is

rejected, then the DR is rejected from the ERB and closed out. If the CM team receives notification that the DR is approved, then the CM team places the DR into the product backlog for consideration for incorporation in the next software sprint. Following execution of adjudication, the CM team generates and provides corresponding meeting minutes for the ERB chair for review. Once the ERB chair provides the final ERB minutes back to the CM team, the CM Team distributes the finalized ERB minutes to the ERB Team, which concludes the ERB process.

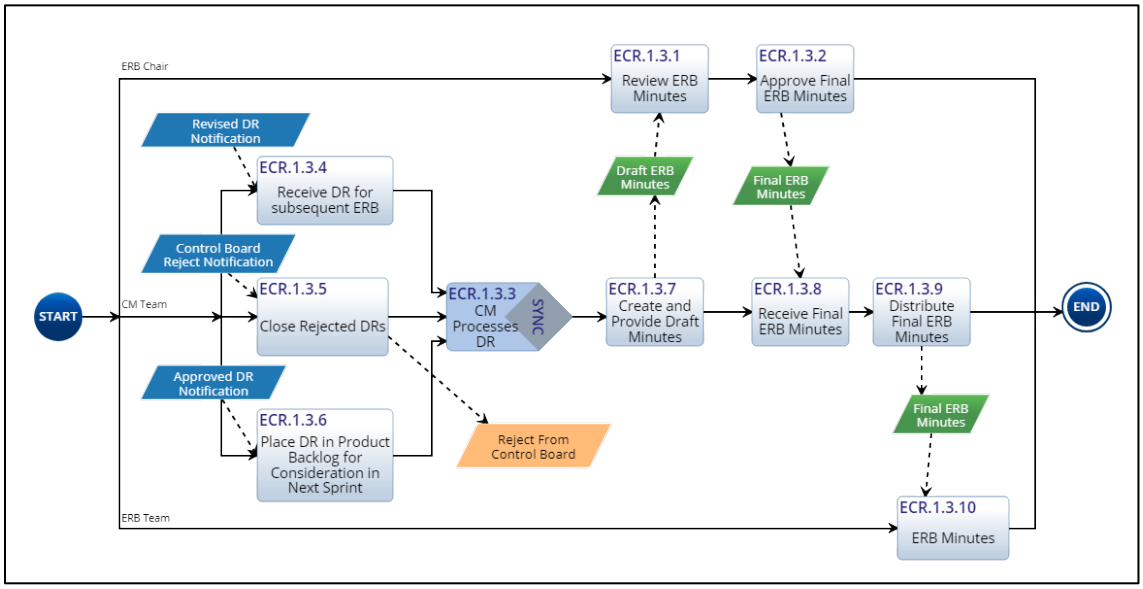



Figure 58. Decomposition of Close ERB (ECR.1.3)

### 7. Perform Continuous Integration and Testing (D.1.5)

The continuous integration and testing process is the last technical action to be performed prior to making a decision to releasing the software. The decomposition of "perform continuous integration and testing (D.1.5)" is performed by the software quality engineering, integration, and test teams (see Figure 59). The action "conduct automated functional testing (D.1.5.1)" is performed by the software quality engineering team (see Figure 60). The software will be loaded on a stand-alone system and the team will develop automated test scripts and conduct automated functional tests of the software. If the testing was successful, the team will proceed to the next action for instrumented and un-

instrumented testing. If automated testing was not successful, the team will determine if work can proceed. If the cause of the failure was a scripting error, the team will correct the syntax and continue with automated testing. If there is a software bug causing issues with the scripting, the team will document the issues and utilize the deficiency report and engineering change request processes. If work cannot proceed after the failure of automated testing, a reject trigger is provided to the top-level "reject or redirect (R.1)" and the process ends.

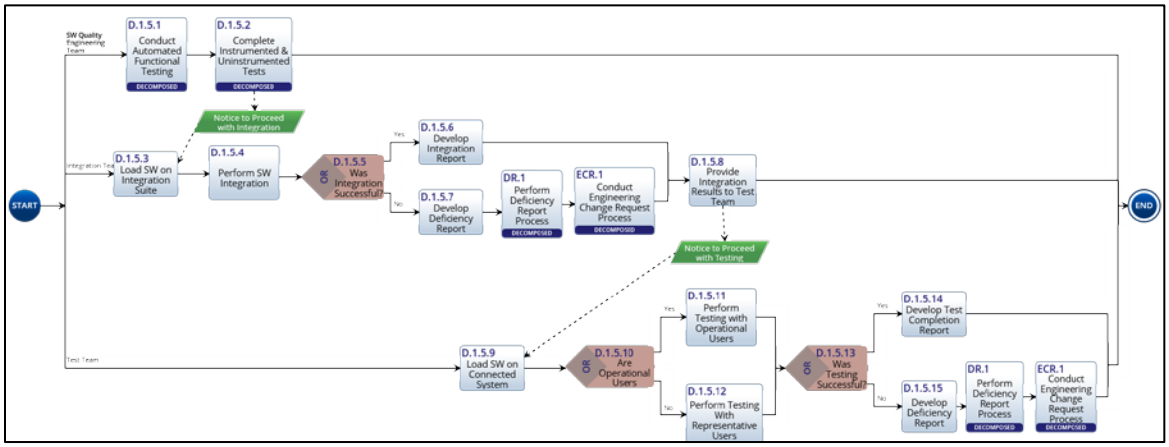



Figure 59. Decomposition of Perform Continuous Testing (D.1.5)

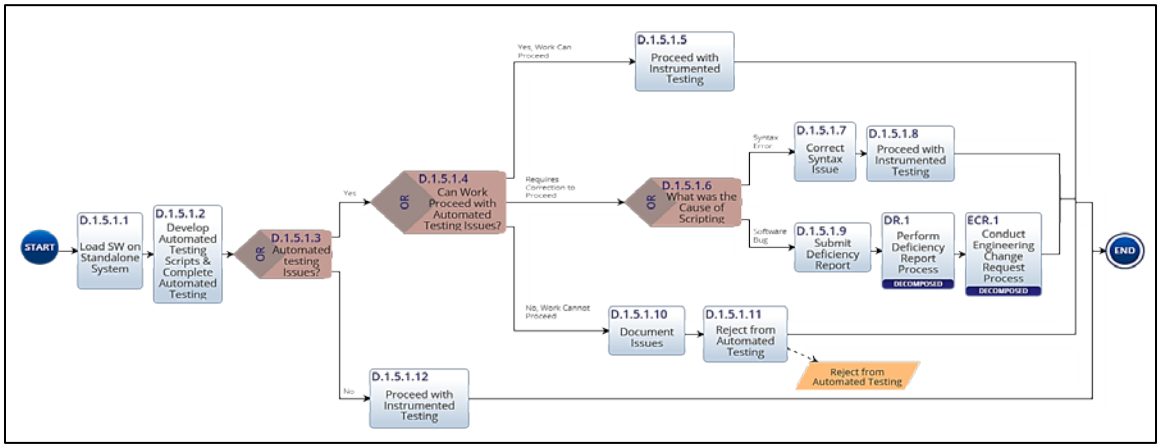



Figure 60. Decomposition of Conduct Automated Functional Testing (D.1.5.1)

The action “complete instrumented and un-instrumented tests (D.1.5.2)” is also performed by the software quality engineering team (see Figure 61). If the software is not already loaded, the team will load it on a stand-alone system and complete instrumented tests. Instrumented testing involves using analytical software tools to detect errors or potential issues in the code. If instrumented testing produced no significant findings, the integration team will be notified to proceed with software integration. If unsuccessful, the software quality engineering team will complete un-instrumented tests. Un-instrumented testing involves manual inspection of the software code. If un-instrumented tests discover significant issues with the software, and work cannot proceed, a reject trigger is provided back to the top-level “reject or redirect (R.1)” and work ends. If issues are minor enough where work can continue, the deficiency report and engineering change request processes will be used, then additional testing efforts will continue.

After functional, instrumented, and un-instrumented tests, the integration team will load or deploy the software to an integration suite. The integration suite may be in a virtualized cloud environment or physical hardware. The integration team will do work necessary to ensure the software successfully works with other system components or external interfaces and is able to deploy in the target environment. If integration is successful, the team will develop and integration report and provide it to the test team. If integration is not successful, the team will develop a deficiency report and utilize the deficiency and engineering change request processes.

Upon completion of integration, the test team will load software on an operationally representative system, connected to the network. Up to this point, all development, integration, or tests have been done in a development environment. The testing on a connected system involves the new software capability, integrated with other system components, and installed on the unclassified or classified network. If operational users are available, they will be offered a chance to perform system testing with other test team members on the project team. If operational users are not available, representative users will perform testing. If successful, a test report will be developed for consideration in the subsequent release decision. If testing was not successful, the deficiencies will be annotated, and the deficiency report and engineering change processes will be enacted.

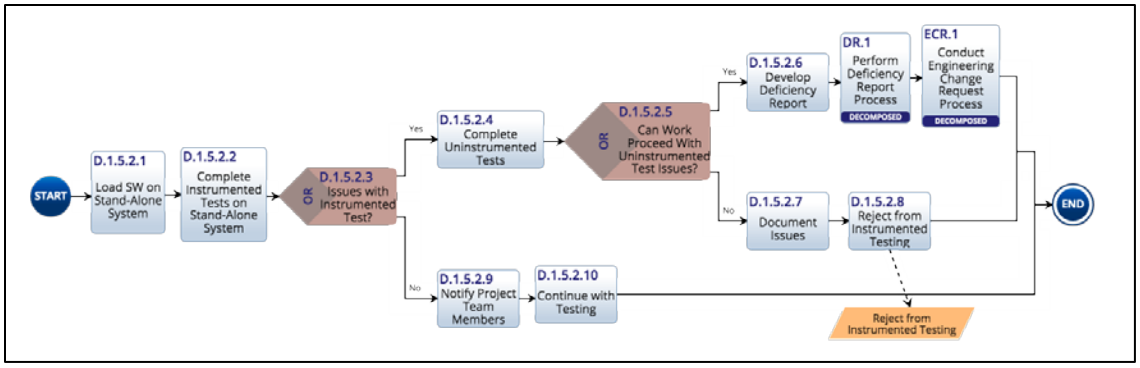



Figure 61. Decomposition of Complete Instrumented and Uninstrumented Tests (D.1.5.2)

### 8. Conduct Build Release Decision (D.1.6)

The last major action is to decide whether or not the software is ready for release. The decomposition of "conduct build release decision (D.1.6)" is performed by the lead systems engineer, project manager, and sponsor (see Figure 62). Taking all prior activities and results into account, the lead systems engineer will provide recommendations to the project manager for releasing the software, including any risks or issues associated. The project manager will liaison with the sponsor to provide a formal recommendation to release the software or not. The sponsor will provide the decision to the project manager, who will disseminate the decision to the lead systems engineer. The scope of this thesis ends with the decision to release the software or to disapprove the release. If the software is approved for release, any follow-on activities can occur. These may include testing to determine whether or not the software is operationally suitable and effective, testing with other external systems, or limited deployment to operational users.

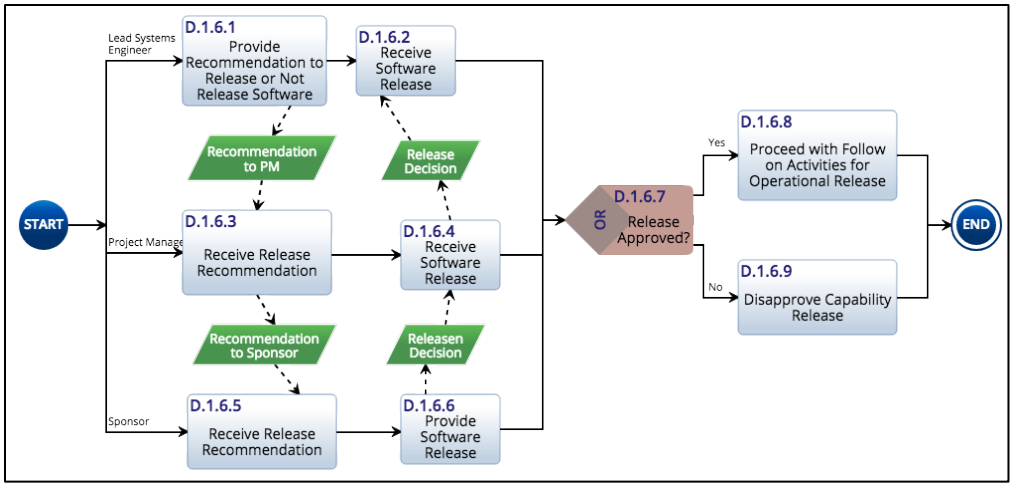



Figure 62. Decomposition of Conduct Build Release Decision (D.1.6)

## 9. Perform Continuous SW Development, Integration, and Test (F.0)

The actions from "perform capability assessment (D.1.1)," "establish templates and verify schedule (D.1.2)," "conduct software sprint (D.1.3)," "perform software quality engineering activities (D.1.4)," "perform continuous integration and testing (D.1.5)," and "conduct build release decision (D.1.6)" were used to develop the decomposed architecture diagram of "perform continuous software development, integration, and test (F.0)" (see Figure 63). As previously discussed, the purpose of F.0 is to execute software sprints and supporting activities that facilitate continuous software design, development, integration, and testing.

There are six main functions and mechanisms within the decomposition. The function "perform capability assessment (F.1)" is performed by the "government assessment team (C.1)." The function "perform software design (F.2)" is performed by the "software design team (C.2)." The function "perform software development and review (F.3)" is performed by the "software development team (C.3)." The function "provide configuration management services (F.4)" is performed by the "configuration management team (C.4)." The function "perform software quality engineering (F.5)" is performed by the "software quality engineering team (C.5)." Lastly, the function "perform continuous integration and testing (F.6)" is performed by the "software integration team (C.6)" (see Figure 63).

The function “perform capability assessment (F.1)” aligns to “perform capability assessment (D.1.1).” If the software is part of an S&T or existing baseline and being proposed for further development, it will undergo an initial capability assessment before being accepted into the design, development, integration, and test pipeline. The output of this function may be an acceptance notification to the software design team, or a capability assessment report back to the business development team with “perform business and technical feasibility analysis (EXT.F.1)” with recommendations to proceed or not proceed (see Figure 63).

The function “perform software design (F.2)” aligns to “establish templates and verify schedule (D.1.2)” and “conduct software sprint (D.1.3).” Refined customer needs, personnel assigned to work, contractual work approval and guidance, and decision to insource work are received as inputs (see Figure 63). Refined customer needs in the form of requirements are used to develop the software roadmap and initial product backlog. These requirements may come from “conduct needs analysis and project planning (EXT.F.1.2),” or they may come from higher level acquisition documents as part of the organizational guidance, government regulations, and industry standards control received by all functions in the IDEF0. Contractual work approval and guidance from “provide post award administration and guidance (EXT.F.2.5)” (see Figure 30) is relevant if the development work will be at least partially performed with a contractor workforce. Conversely, the decision to insource work from “accept, redirect, or reject work (EXT.F.1.3)” signals that the effort will be done internally without assistance from contractors (see Figure 19). This function also receives additional inputs as feedback loops from “perform software development and review (F.3)” for software sprint review data and any product backlog updates. Sprint review data and updates to the product backlog are used for planning follow on sprints. “Perform software quality engineering (F.5)” provides inputs to F.2 for software reject notifications from internal government developers and software change requests. When contractors are used, the reject notification will flow back to “provide post award administration and guidance (EXT.F.2.5).” The function “provide continuous integration and testing (F.6)” also provides inputs to “perform software design (F.2)” for software reject notifications, user feedback, and software change requests.

The function “perform software development and review (F.3)” aligns to “conduct software sprint (D.1.3).” The software design team via “perform software design (F.2)” will provide a software release plan and initial product backlog to the software development team. The software development team will provide software deliveries to the configuration management team as software development progresses. As previously mentioned, the development team also provides sprint review metrics back to the software design team along with any updates to the product backlog. In the same manner as software design, when contractors are used, reject notifications are provided as an input to “provide post award administration and guidance (EXT.F.2.5).”

The function “provide configuration management services (F.4)” aligns to “establish templates and verify schedule (D.1.2)” and “conduct software sprint (D.1.3).” Software deliveries are processed through the CM team to maintain configuration management of software baselines. After delivery, the development team will retrieve code from CM and return code back to CM after modifying to ensure the software is configuration managed properly. Software is also provided as an output for use by the integration and test team to “perform continuous integration and test (F.6).” Software delivery info as an output to “perform market research and develop contracting materials (EXT.F.2.2)” (see Figure 30). This information helps the contracting team in writing CDRLs and other task order delivery specifications to include in the contracting language. This function also receives inputs as feedback loops from “perform software quality engineering (F.5)” and “perform continuous integration and test (F.6).” Executable software, software change requests, and software reject notifications are provided from F.5 to F.4. Integrated and tested software, software change requests, and reject notifications are provided from F.6 (see Figure 63).

The function “perform software quality engineering (F.5)” aligns to “perform software quality engineering activities (D.1.4).” Personnel assigned to work and CM baselined source code are provided as inputs. In addition to the feedback loops from F.5 provided to F.4 as inputs, F.5 also provides a feedback loop of software rejection notifications to “perform software design (F.2)” (see Figure 63).

The function “provide continuous integration and testing (F.6)” aligns to “perform continuous integration and testing (D.1.5).” The inputs for this function include personnel assigned to work and software for integration and testing from the CM team. The outputs from this function were previously discussed as feedback loops to “provide configuration management services (F.4)” and “perform software quality engineering (F.5).” In addition, the feedback loop for software change requests is also provided to “perform software design (F.2).”

Lessons learned and software acceptance or rejection notifications are provided as outputs from all functions within architecture IDEF0 of “perform continuous software development, integration, and test (F.0),” except for “perform capability assessment (F.1).” The purpose of omitting software acceptance or rejection notifications from F.1 is intentional. Software rejections or acceptance from contractors has to work through formal contracting mechanisms; whereas government deliveries do not require the same level of formality. The model intentionally segregates rejection notifications for government and contractor software deliveries. Initial capability assessments are performed on software capabilities that are not yet accepted into the design, development, integration, and test cycle. As such, issues with capability assessments can still result in a rejection of software; however, this would be done via a capability assessment report back to some level of organizational leadership or business development to make a final determination.

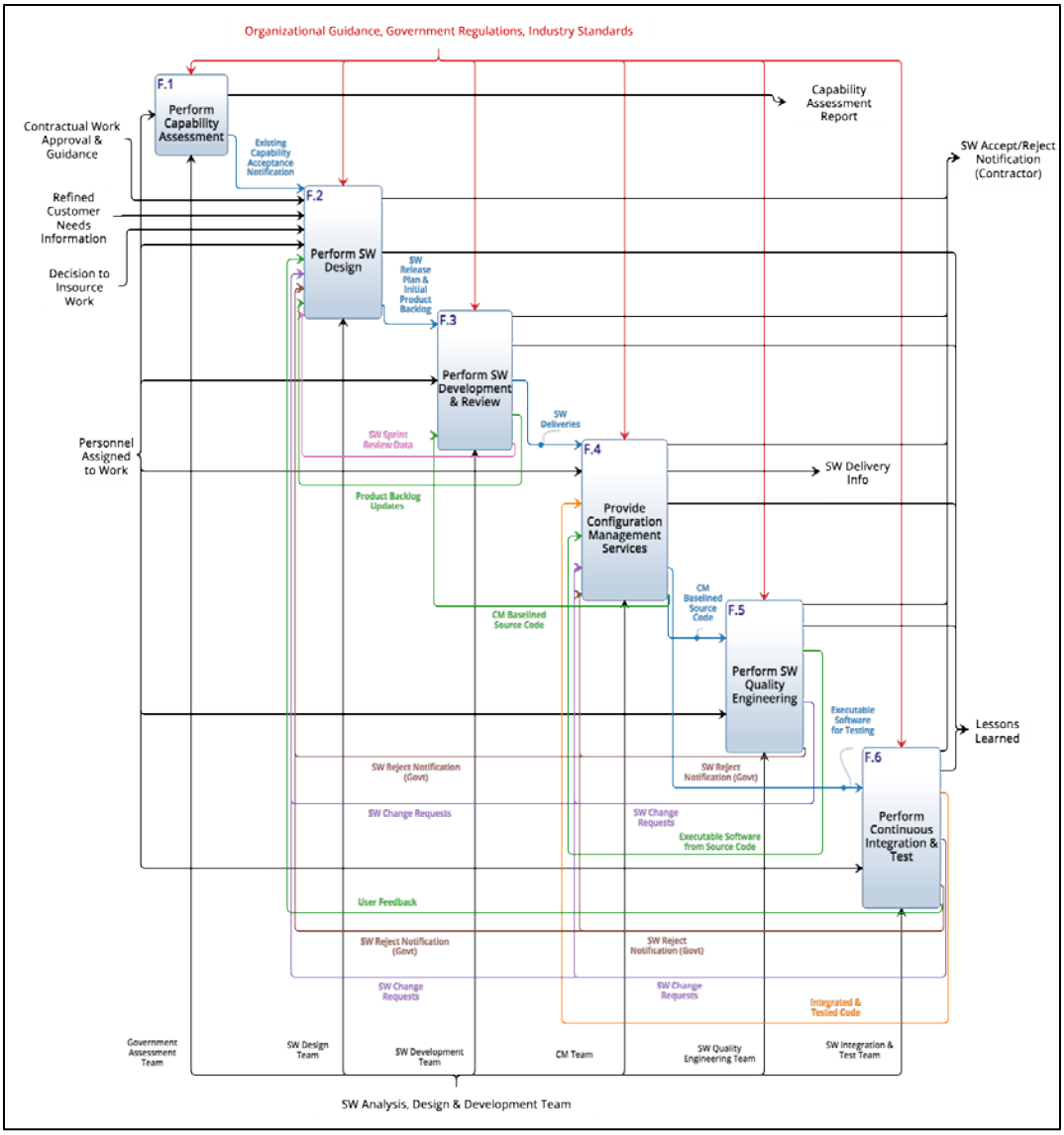



Figure 63. Architecture IDEF0 of Perform Continuous SW Development, Integration, and Test (F.0)

## G. CHAPTER CONCLUSIONS

This chapter discussed action and IDEF0 diagrams developed as part of an analysis of existing organizational SOPs for agile software development. The action diagrams were used to develop the architectural IDEF0 diagrams. Action diagrams were decomposed to the lowest level needed to support creation of architectural diagrams. Chapter III Section B discussed the decomposition of the level 0 action and IDEF0 models. Chapter III Section C discussed models for business and technical feasibility analysis. Chapter III Section D discussed models for assessing organizational personnel to perform the work associated with software development. Chapter III Section E provided a decomposition of the contract task order process within the organization. Chapter III Section F discussed the modeling and decomposition of agile software development.

A finding from this model development is that the current SOPs adequately cover contracting for software development, pre-vetting and capability assessments of S&T based software, and processes for continuous integration and testing. Current SOPs do not cover the agile software design and develop actions followed by existing projects within the organization. The following action diagram hierarchy can be used to develop new SOPs for the organization: “conduct software sprint (D.1.3),” “perform initial backlog and sprint planning (D.1.3.1),” “perform sprint planning, execution and review (D.1.3.3),” and “receive software delivery and documentation (D.1.3.4).” There is also no SOP for personnel assessment, currently done in an ad hoc fashion. The organization may benefit from adopting a more structured approach to personnel assessment for soliciting interest for new opportunities and assigning employees to those efforts. Technical and feasibility assessments are performed using a consistent and structured approach, which is correctly captured as part of the process architecture. Additionally, internal contracts processes are also correctly captured as part of the process architecture but additional efforts can be undertaken to collect historical data for further analysis and model refinement. In fact, collection of historical data is pertinent for all actions within the process architecture.

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. MODEL USAGE

## A. INTRODUCTION

Chapter III discussed the action diagrams and modeling used to develop an architecture for agile software development. In Chapter IV, a subset of those action diagrams was used to perform effort and cost-based Monte Carlo simulations specifically for the action, "conduct software sprint (D.1.3)." The scope of the simulations is limited to data from a single software development project within the C2IS division. Section B discusses model constraints and how the values used for simulation were obtained. Section C discusses the construct of the actions used to support the Monte Carlo simulation for "perform initial backlog and sprint planning (D.1.3.1)." Section D discusses the actions used in support of the Monte Carlo simulation for "perform sprint planning, execution and review (D.1.3.3)." Section E discusses the actions used and resulting the Monte Carlo simulation for "receive software delivery and documentation (D.1.3.4)." Section F discusses the Monte Carlo simulation for all three major actions contained within "conduct software sprint (D.1.3)." These include "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)." Section G provides a model use case to demonstrate model usage. Lastly, Section H discusses the chapter conclusions based on the numerous Monte Carlo simulations.

## B. MODEL CONSTRAINTS FOR SIMULATION

Once the models were complete, then validated by subject matter experts for their respective area within the organization, we assigned values to actions within the model to simulate effort and cost. Model input parameters included the number of personnel roles, hourly pay rate, and allocated hours for each personnel role. Model simulation outputs provide the effort and cost with the mean and standard deviation for 1000 Monte Carlo trials (see Figure 64). Monte Carlo simulations allow for multiple trial runs of large sample sizes. "Monte Carlo is a 'game of chance' technique used to solve many types of problems by applying random sampling instead of analytic methods" (Madachy and Houston

2018,75). Within the C2IS division, the model's output of Monte Carlo simulations can be used by management to assess effort and cost estimation risk based on randomness within model parameters. The simulated output can be used by project managers to ensure effort and cost estimates for initial backlog and sprint activities provided by the software development team are reasonable and within a risk tolerance determined by management.

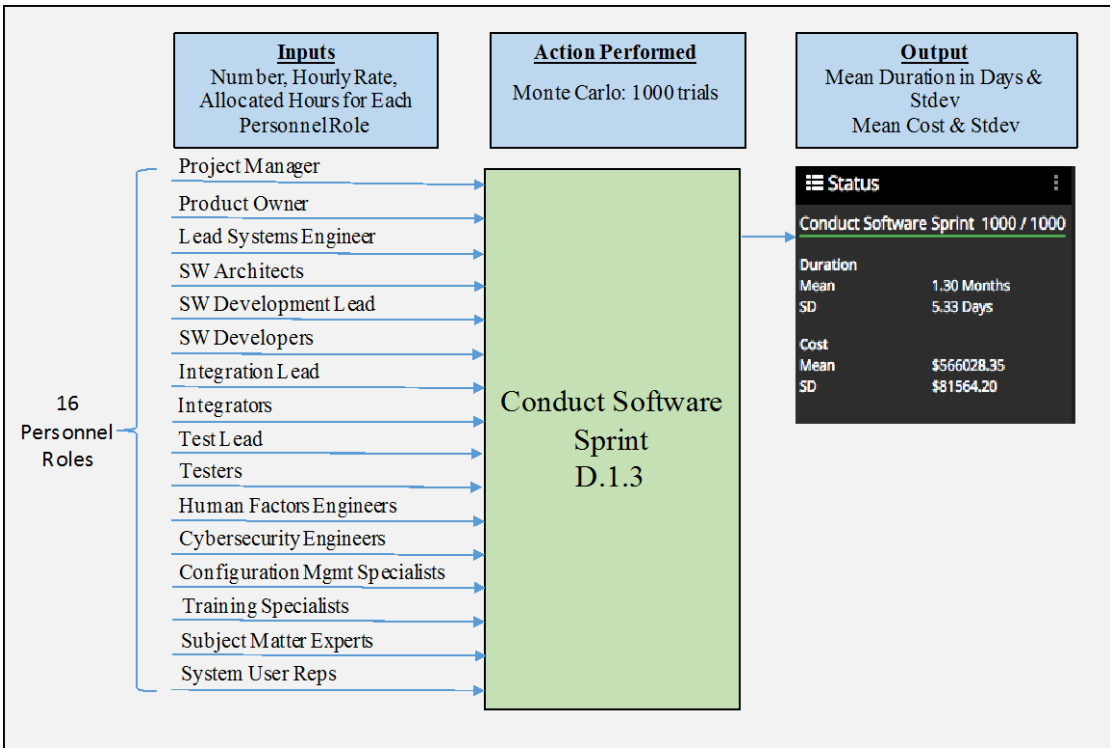



Figure 64. Model Input Parameters and Outputs for Conduct Software Sprint (D.1.3)

Values within each action are either duration based using hours, or they are based on a normal distribution with mean and standard deviation (see Figure 65). The attributes for each item that incurs cost is defined separately. Innoslate allows for multiple cost attributes to be assigned to one or more actions within the entity view of each action diagram, under the program management tab of relationships (see Figure 66).

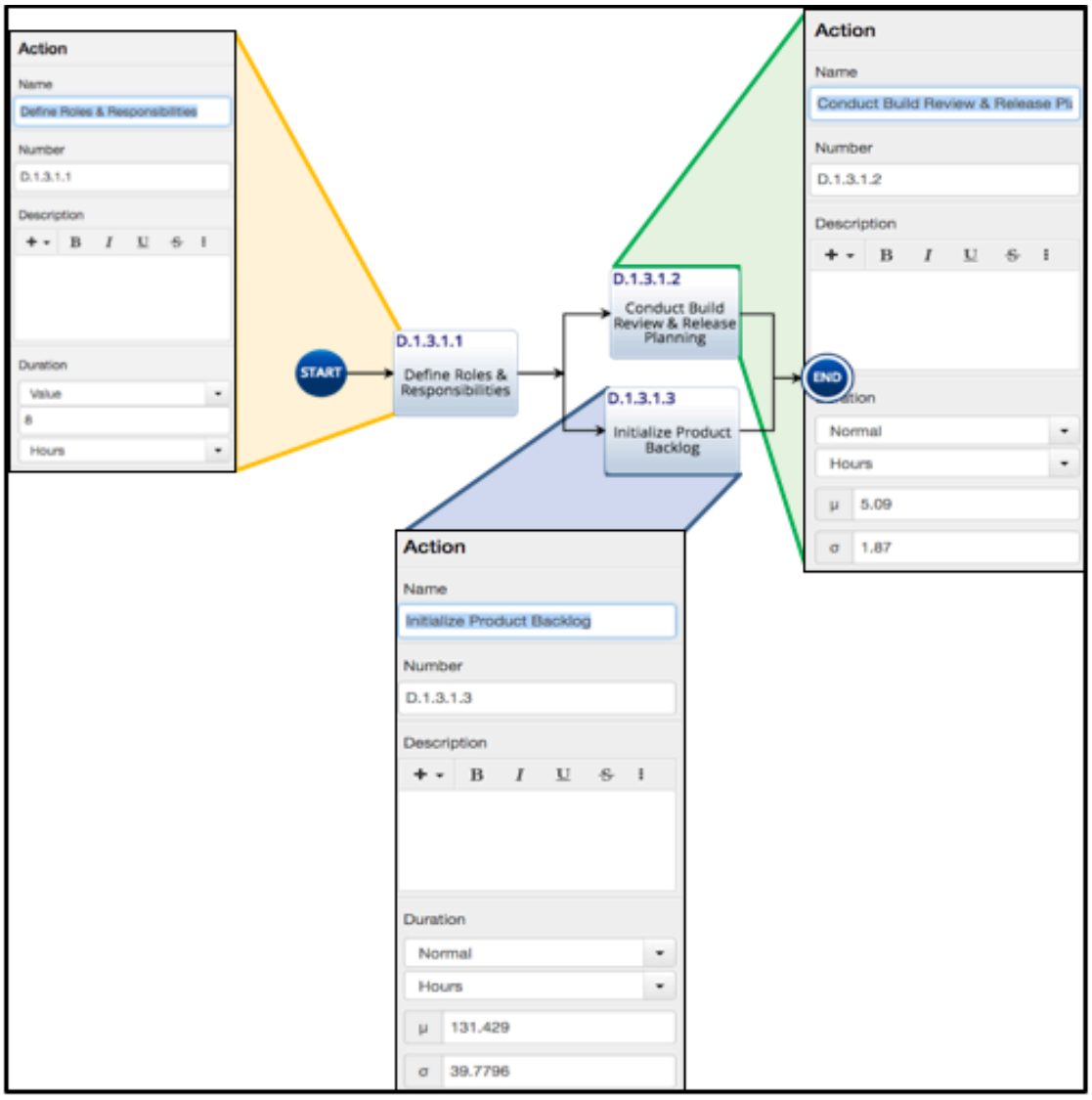



Figure 65. Action Durations for D.1.3.1

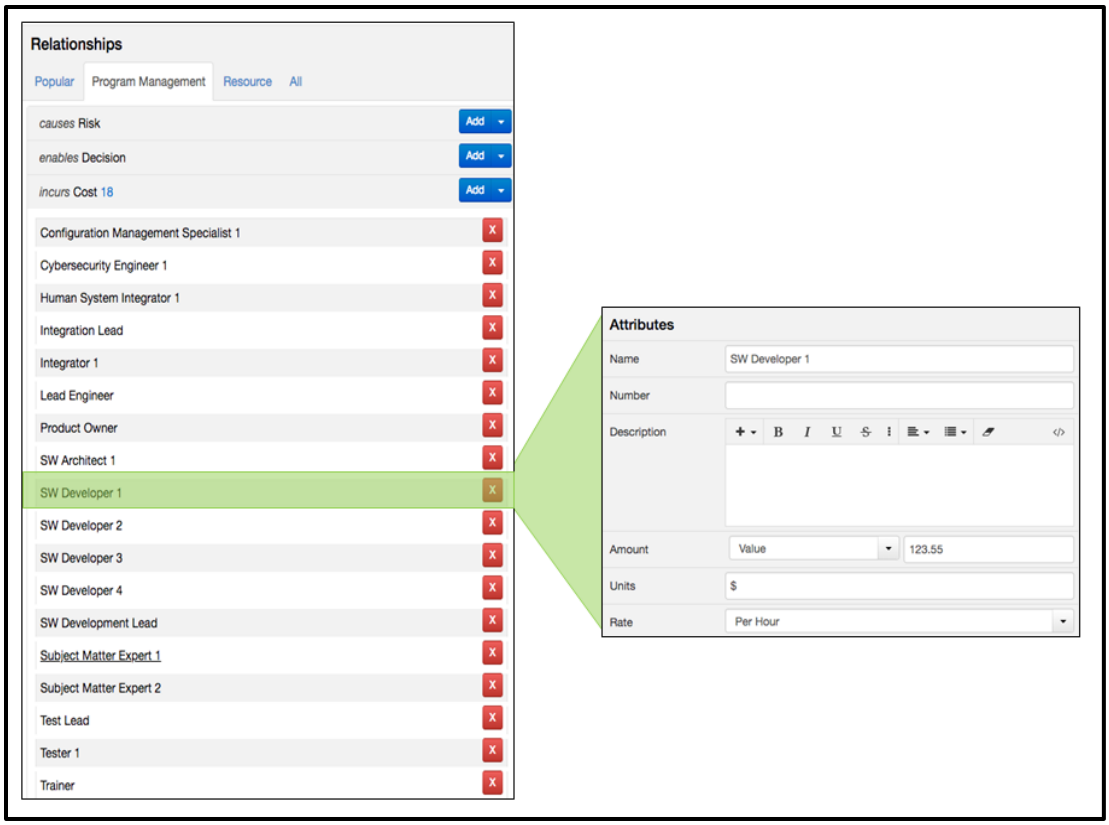



Figure 66. Incurs Cost for Initialize Product Backlog (D.1.3.1.3)

After the decomposition of all actions within the A.0 model (see Figure 13 in Chapter III), we determined that there was insufficient data available within the C2IS division for use in the simulation actions other than "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)." Once the modeling was complete, we analyzed available data and applicable fit within the model for simulation. Data captured for the simulation includes actual work hours per developer per sprint for 10 sprints, and actual cost per sprint for seven sprints. However, the data for supporting actions leading up to and after software development (D.1.3) were not available. Durations for software development are based on the mean and standard deviation of allocated development hours per developer within 10- or 15-day software sprints. The model is able to scale based on the time duration of software sprints to provide dynamic output. This includes the number and type of personnel, as well as the fully burdened hourly rate for the

C2IS division in fiscal year 18 (see Table 1). Data for pre, and post sprint activities was provided by subject matter experts in the areas of initial backlog and sprint planning, sprint reviews, and configuration management of software deliveries. The entire data set used to support model development can be found in Appendix A. Metrics utilized throughout this thesis were generalized relative to employees within the C2IS division. This thesis does not collect or include information attributable to specific individuals. Due to the use of generalized hourly rates versus using employee actual rates, the model simulation outputs for cost is not as precise as it could be if actual employee hourly rates were used.

Model design is intentionally modular. The actions "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)" can be simulated as a whole, or individually. This methodology provides a foundation and experimental test bed for future modeling and can be scaled for various sized software projects within the C2IS division.

Table 1. Personnel Roles and Hourly Rates

| Role | GS-Equivalent | Hourly Rate (FY18 burdened) |
|---|---|---|
| Project Manager | GS-15 | $154.14 |
| Product Owner | GS-13 | $126.50 |
| Lead Engineer | GS-13 | $123.55 |
| Architecture | GS-13 | $123.55 |
| Software Development Lead | GS-15 | $154.14 |
| Software Developers | GS-13 | $123.55 |
| Integration Lead | GS-13 | $123.55 |
| Integrators | GS-11 | $108.08 |
| Test Lead | GS-13 | $126.50 |
| Testers | GS-11 | $108.08 |
| Human Systems Interface (HSI) | GS-13 | $123.55 |
| Cybersecurity | GS-13 | $126.50 |
| Configuration Management (CM) | GS-11 | $108.08 |
| Trainers | GS-11 | $108.08 |
| System User | GS-11 | $108.08 |
| Subject Matter Experts (SMEs) | GS-13 | $123.55 |

## C. PERFORM INITIAL BACKLOG AND SPRINT PLANNING (D.1.3.1)

To perform the Monte Carlo simulation, we assigned values to each decomposed action within "perform initial backlog and sprint planning (D.1.3.1)" as shown in Figure 67. No values were assigned to the action of "perform initial backlog and sprint planning (D.1.3.1)." Instead, values were assigned individually to the decomposed actions for "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)."

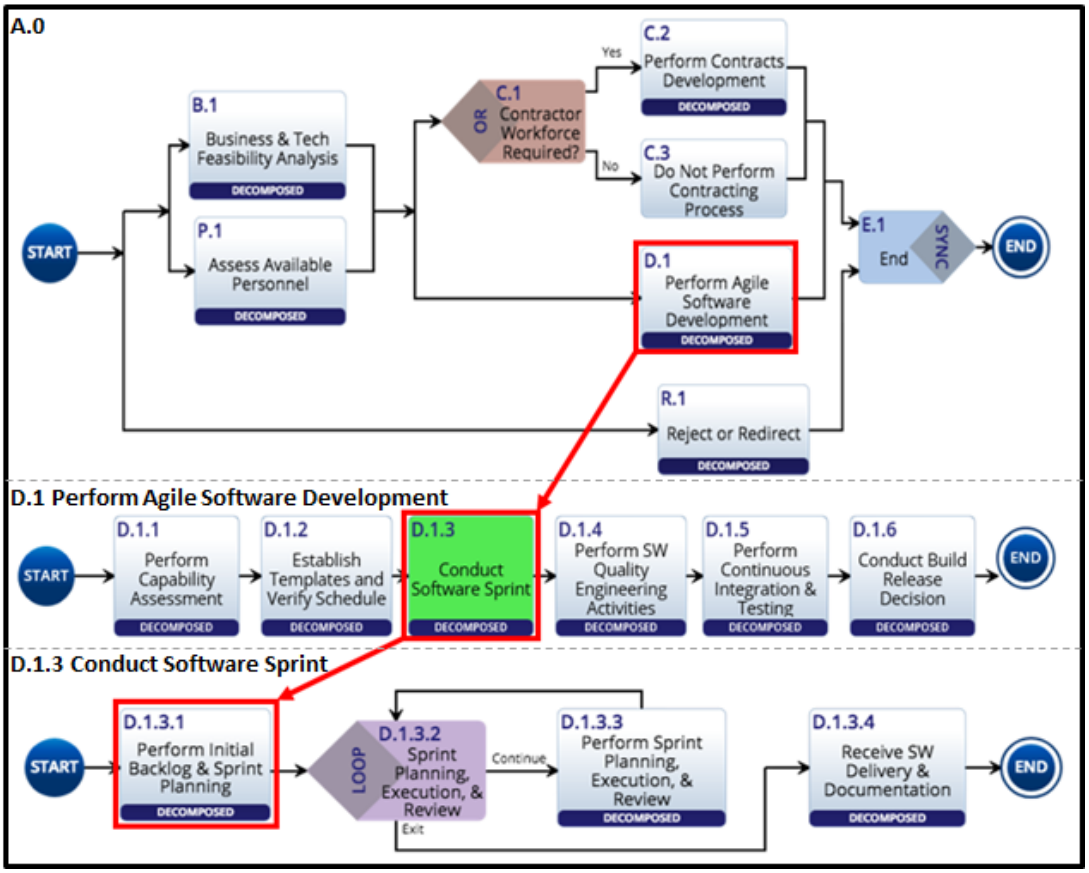



Figure 67. Perform Initial Backlog and Sprint Planning D.1.3.1

The duration values or statistical data assigned to each action are used within the Monte Carlo simulation to forecast effort and cost. Since there are no input triggers provided to the first action, "define roles and responsibilities (D.1.3.1.1)," the Monte Carlo

simulation can be performed. If there were an input trigger to D.1.3.1.1, a discrete simulation would be required. This is an example of modularity within the model design (see Figure 68).

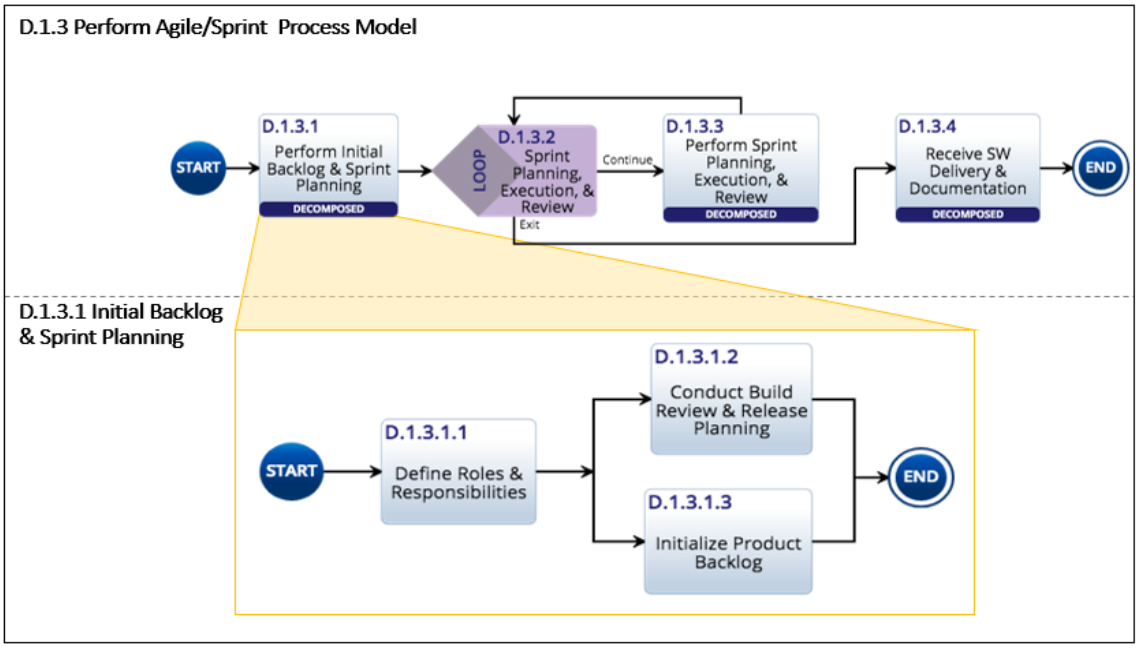



Figure 68. Decomposition of Perform Initial Backlog and Sprint Planning (D.1.3.1)

A Monte Carlo simulation with 1000 trials of “perform initial backlog and sprint planning (D.1.3.1)” resulted in an average duration of 18.2 days with a standard deviation of 4.77 days. The average cost via simulation was $304,395 with a standard deviation of $81,000 (see Figure 69). The Y axis shows the number of simulations and the X axis shows total cost.

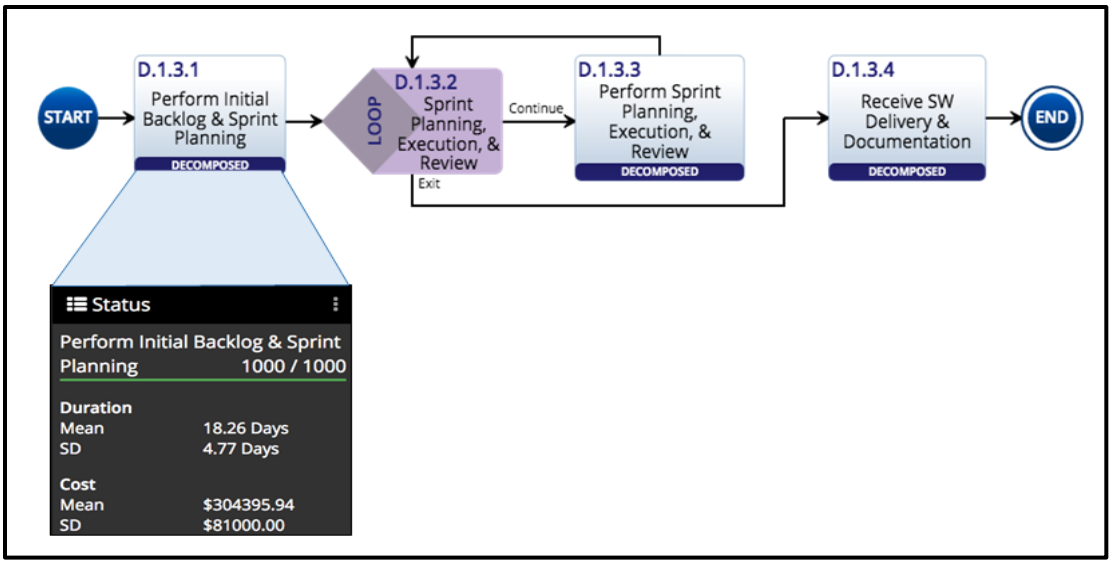



Figure 69. Monte Carlo Simulation 1000 Trials for Perform Initial Backlog and Sprint Planning (D.1.3.1)

Based on a single 1000 trial run of the Monte Carlo simulation, 585 out of 1000 trials (58.5%) resulted in a cost ranging from $245,605–$313,313 (see Figure 70). This simulation was performed using the 15-day sprint effort with 15 personnel roles assigned. This is calculated by adding the number of simulations in the two middle bars within the (see Figure 70). Additionally, 760 out of 1000 trials (76%) resulted in cost that was greater than $245,605. This is calculated by adding the number of simulations from the $245,605 bar to the right (see Figure 70). Thus, a manager can confidently state that 76% or more of the time, the cost for this set of actions will be greater than $245,605. Additionally, 82% of the time the cost will be less than $381,021. This is calculated by adding the number of simulations from the $381,021bar to the left (see Figure 70). A manager can confidently state that 82% or less of the time, the cost for this set of actions will be less than $381,021. Managers can use the results such as these to make risk adjusted decisions based on his or her tolerance of risk pertaining to cost estimations.

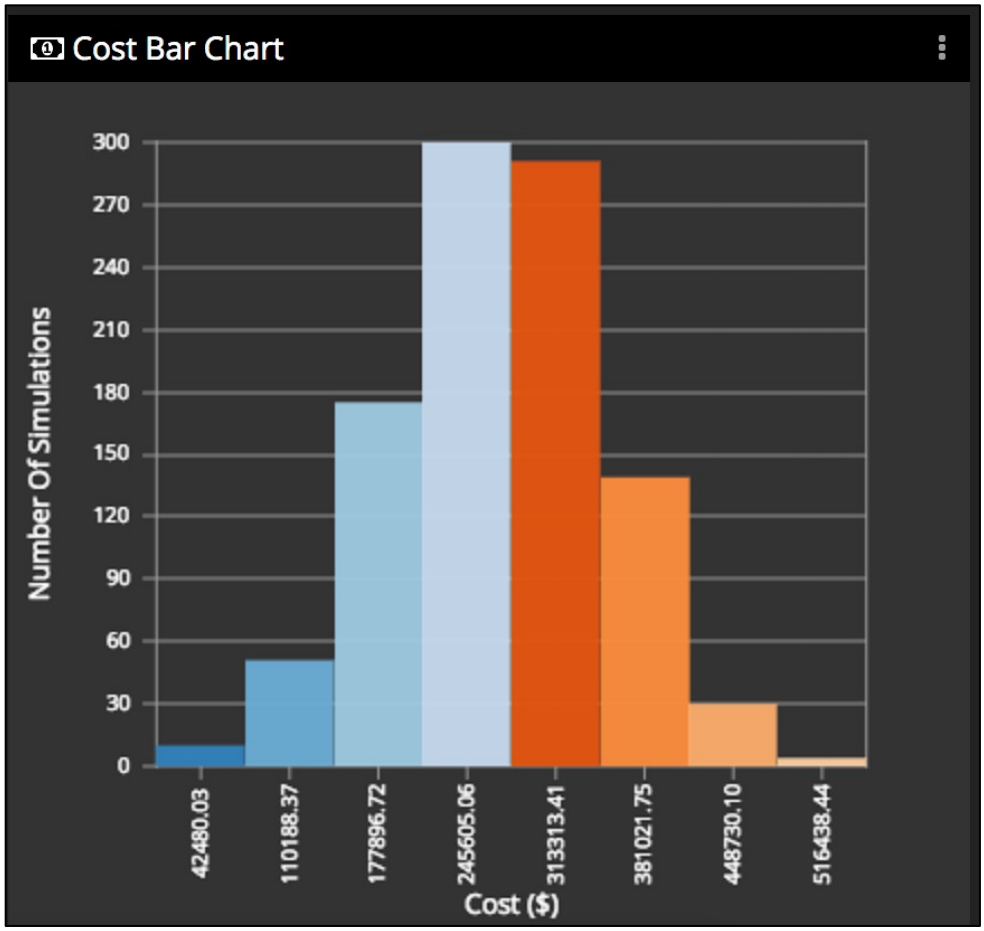



Figure 70. Monte Carlo Cost Simulation with 1000 Trials for Perform Initial Backlog and Sprint Planning (D.1.3.1)

Based on the simulation, 600 out of 1000 trials (60%) resulted in a total effort of 15.1–19.1 days; additionally, 900 out of 1000 trials (90%) resulted in a total effort of 11.1–23.1 days. For approximately 75% of the effort, the number of days to complete "perform initial backlog and sprint planning (D.1.3.1)" will be 15 days or greater. This is found by adding the number of simulations from 15.1 days to the right (see Figure 71). Additionally, for 83% of the effort, the number of days required will be less than 19 days. This is found by adding the number of simulations from 19.1 days to the left. The average of 18.2 days seems reasonable; however, since 83% of the effort the number of days required is less than 19 days, there is a greater risk of over-estimating effort required to perform these actions. Gathering metrics over time and across multiple projects will facilitate more accurate effort estimates based on analysis of Monte Carlo simulation results. Managers

can use the results such as these to make risk adjusted decisions based on his or her tolerance of risk pertaining to estimations for effort.

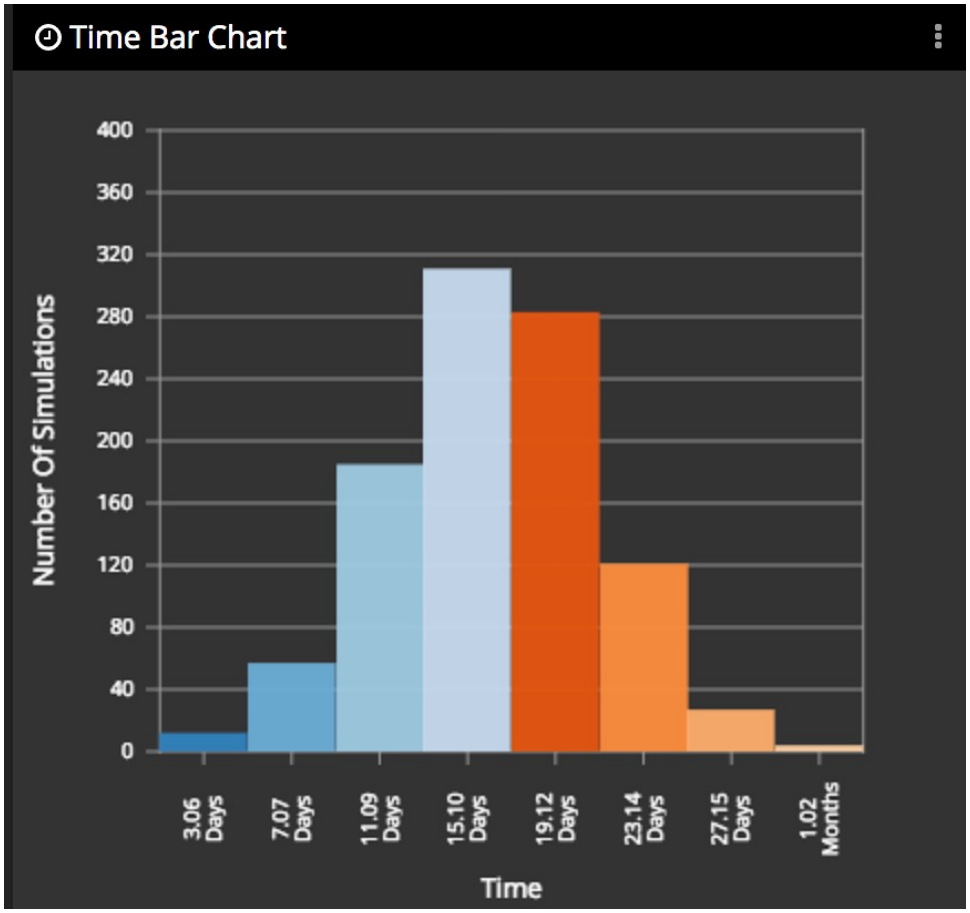



Figure 71. Monte Carlo Time Simulation with 1000 Trials for Perform Initial Backlog and Sprint Planning (D.1.3.1)

## D. PERFORM SPRINT PLANNING, EXECUTION AND REVIEW (D.1.3.3)

To perform the Monte Carlo simulation for "perform sprint planning, execution and review (D.1.3.3)," we assigned values to each decomposed action within D.1.3.3 (see Figure 72). As with the previous simulation, no values were assigned to the higher-level D.1.3.3 action. Values were assigned to actions within "perform sprint planning, execution and review (D.1.3.3)" using a bottom-up approach. See Appendix A for the values assigned to decomposed actions in Figure 72, Figure 73, and Figure 74.

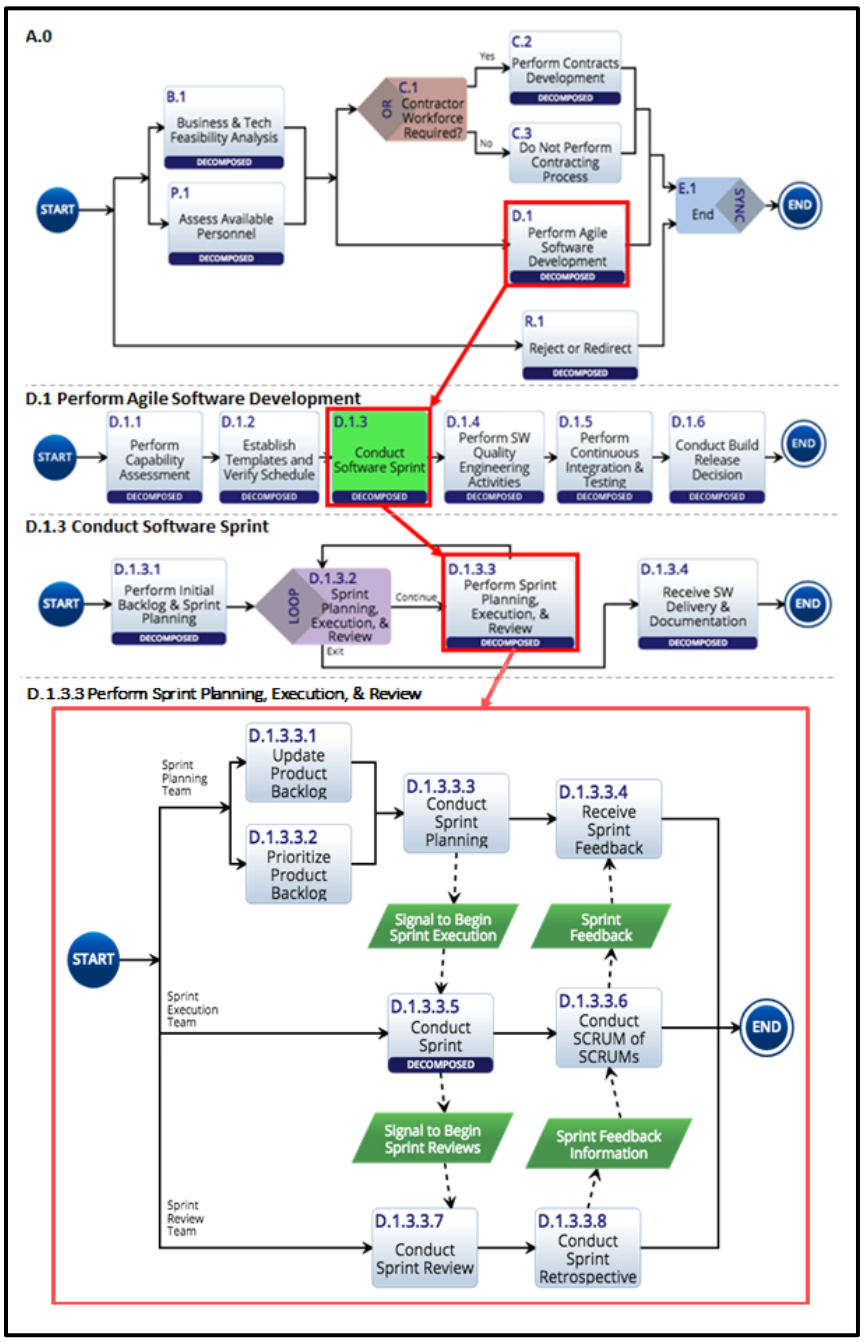



Figure 72. Decomposition to D.1.3.3

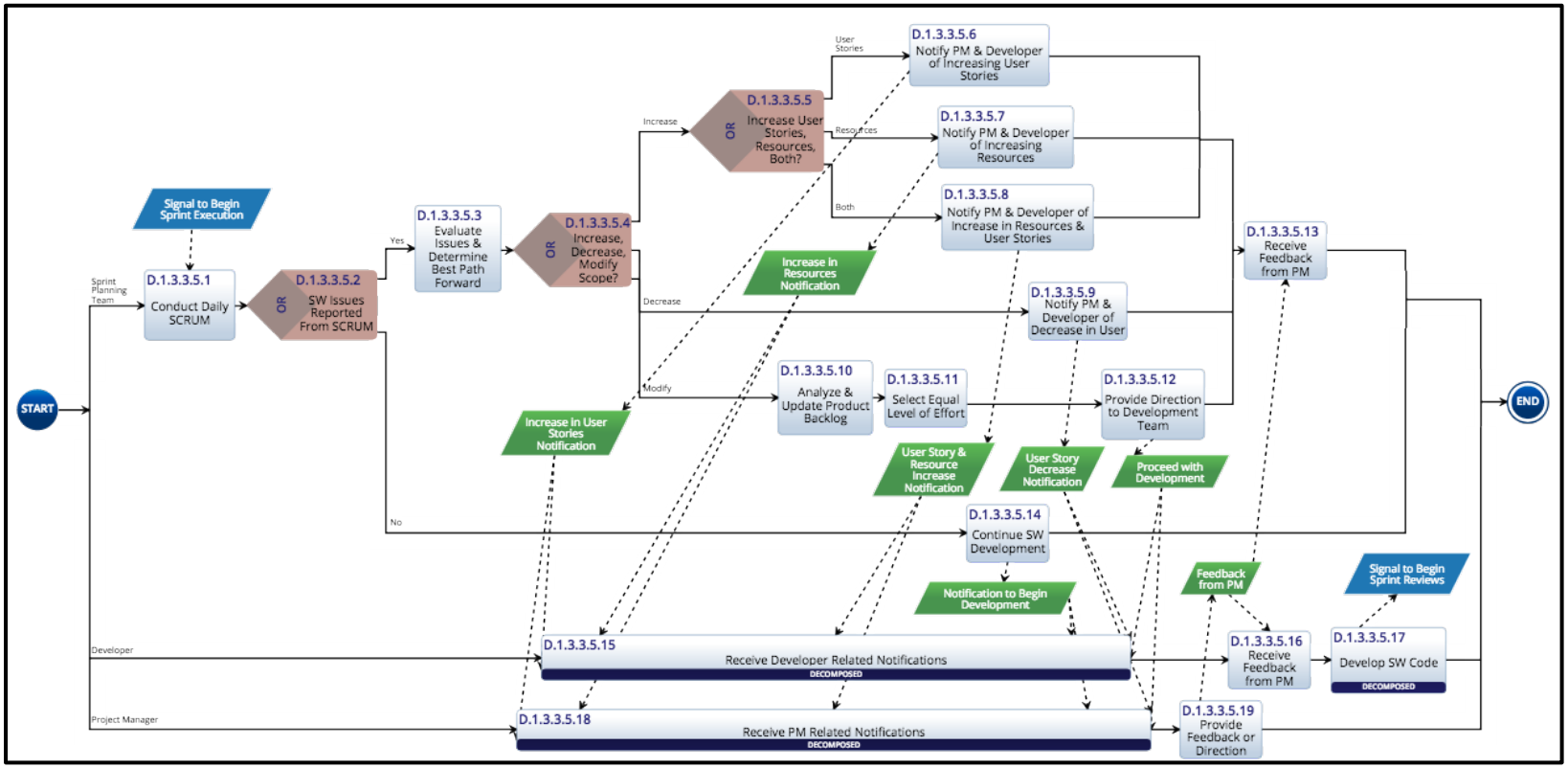



Figure 73. Decomposition of Conduct Sprint (D.1.3.3.5)

Within the “conduct sprint (D.1.3.3.5)” action, the simulation of “develop software code (D.1.3.3.5.17)” utilizes historical sprint metrics from a C2IS division software development project. The actions and effort-based metrics that are involved in develop software code are shown in Figure 74. The process of developing software code is contained within a loop to provide a means to emulate the number of days the sprint is conducted. As previously discussed, sprints for the software project within the C2IS division are time-bound to 10 or 15 days. When the sprint cycle is 15 days, the “LOOP” action for “develop software code (D.1.3.3.5.17.1)” is set to 14 iterations, since the action occurs once and then repeats 14 times. When the sprint cycle is 10 days, the loop iterations is set to nine. The actions that take place within the software development process include, “develop software code (D.1.3.3.5.17.2),” “complete peer review of delivery (D.1.3.3.5.17.3),” and “commit code to software repository (D.1.3.3.5.17.4),” which then repeat as specified by the “LOOP” function.

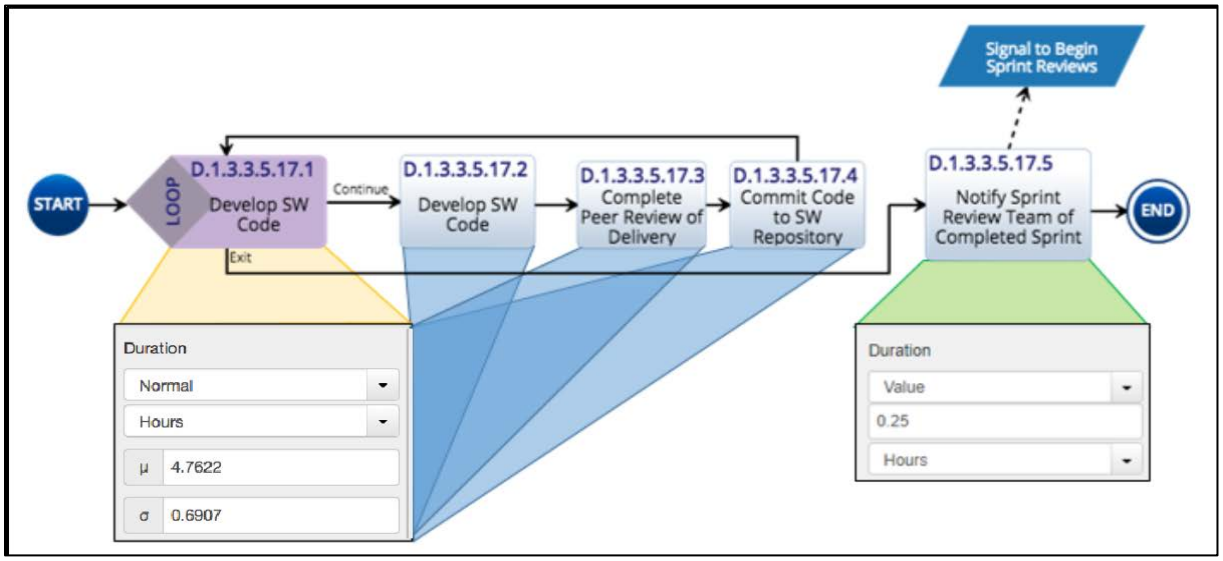



Figure 74. Decomposition of Develop Software Code (D.1.3.3.5.17)

Historical data from 10 prior sprints provided in Table 2 is used to calculate the values illustrated in Figure 74. For a sprint period of 15 days, the “develop software code (D.1.3.3.5.17.1)” activity is allocated an average of 4.7622 hours and standard deviation of 0.6907 hours. When the sprint duration is 10 days, the average used is 4.8867 hours with a

standard deviation of 0.6376 hours. Only the “LOOP” action in Figure 74 is assigned this duration, which incorporates the durations associated with activities within the loop. While the normal work day encompasses eight hours, on average the developer was productive at generating code for 4.7662 hours. Thus, in the case of a 15-day sprint cycle, the results of the simulation can greater or less than 15 days based on the Monte Carlo simulation.

Table 2. Sprint Metrics for Develop Software Code (D.1.3.3.5.17)

| Sprint Number | Number of Days in Sprint | Total Actual Hours | Number of Developers | Average Person-Hours/ day | Standard Deviation |
|---|---|---|---|---|---|
| 1 | 10 | 781 | 17.5 | 4.89 | 0.64 |
| 2 | 10 | 1104 | 20 | | |
| 3 | 10 | 1038 | 20 | | |
| 4 | 10 | 912.5 | 20 | | |
| 5 | 10 | 929 | 20 | | |
| 6 | 15 | 1430.5 | 17 | 4.76 | 0.69 |
| 7 | 15 | 950 | 16 | | |
| 8 | 15 | 1217 | 16.5 | | |
| 9 | 15 | 1400 | 22 | | |
| 10 | 15 | 1314 | 20.5 | | |

The Monte Carlo simulation model produces cost/schedule probability distributions. Those distributions are essentially estimation models to be used on sprints. Estimators/managers can look at the cumulative distribution function (CDF) and gauge their risk. This probabilistic estimation approach is different than using parametric cost estimation models. The results of a Monte Carlo simulation with 1000 trials is shown in Figure 75. From the simulation of D.1.3.3, the conclusion can be drawn that it will take an average 15.9 days to complete the sprint planning, execution, and review process with a standard deviation of 2.6 days. The average cost output is $254,434 with a standard deviation of $20,131.

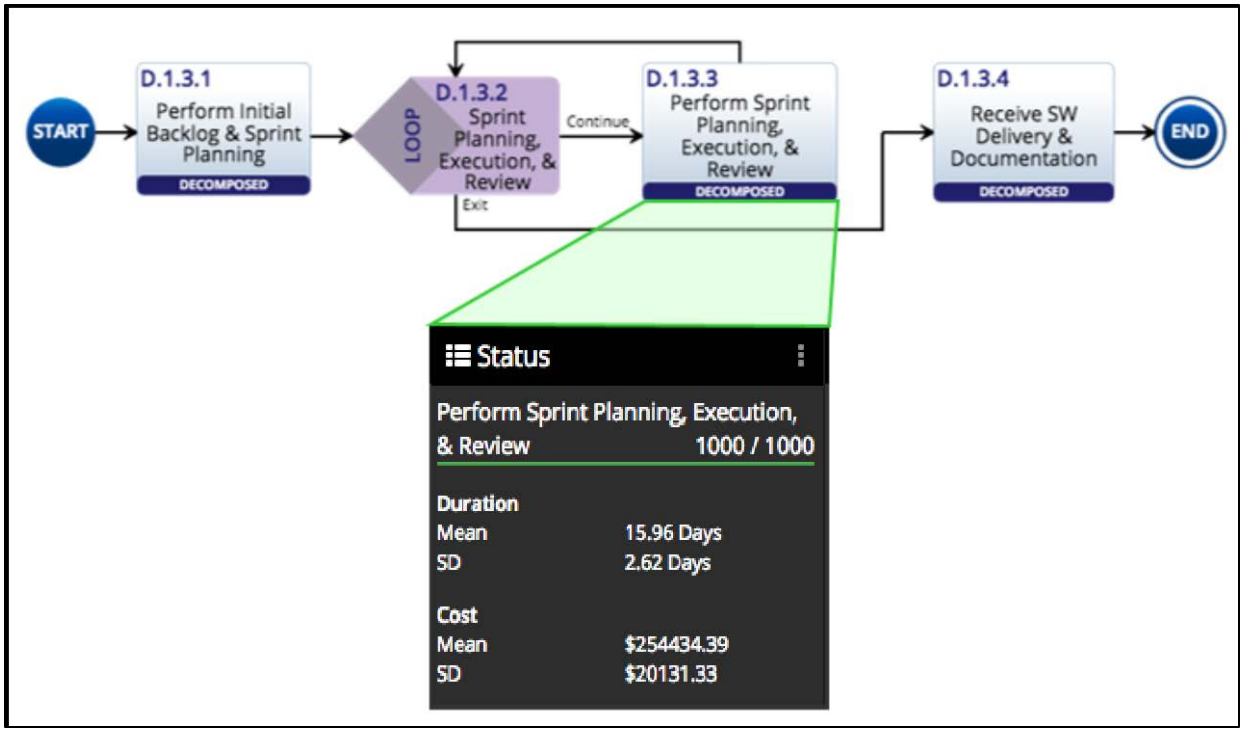



Figure 75. Monte Carlo Simulation Summary with 1000 Trials for Perform Sprint Planning, Execution and Review (D.1.3.3)

To provide additional insight regarding the cost of a 15-day simulation, Figure 76 provides a bar chart showing the cost values returned for a 1000 iteration Monte Carlo simulation. The Y axis provides the number of simulations conducted for D.1.3.3 and the X axis provides the cost in dollars. This chart supports the conclusion that the average cost to perform sprint planning, execution and review is $254,434 (see Figure 76). This Monte Carlo simulation shows a normal distribution with data aggregating around the mean. By adding the bars for number of simulations from $255,035 to the left, we can determine that for 75% of the effort, cost will be less than or equal to $255,035. Adding the bars for the number of simulations from $225,959 to the right, we can establish that for 90% of the effort, costs will be greater than or equal to $225,959. Managers can use the results such as these to make risk adjusted decisions based on his or her tolerance of risk pertaining to estimations of effort. In good practice, an estimate of $225K to $255K would be an acceptable level of risk.

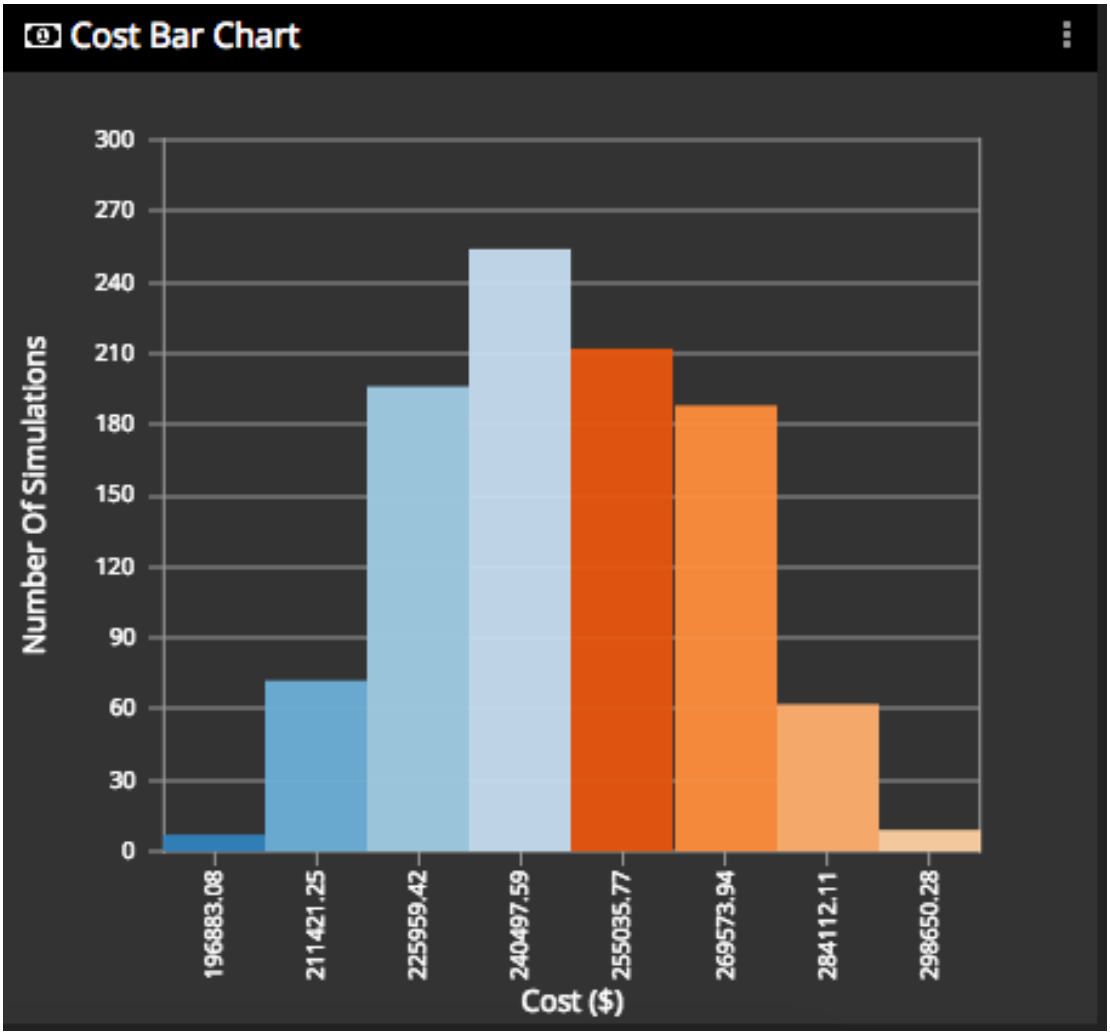



Figure 76. Monte Carlo Cost Simulation with 1000 Trials for Perform Sprint Planning, Execution and Review (D.1.3.3)

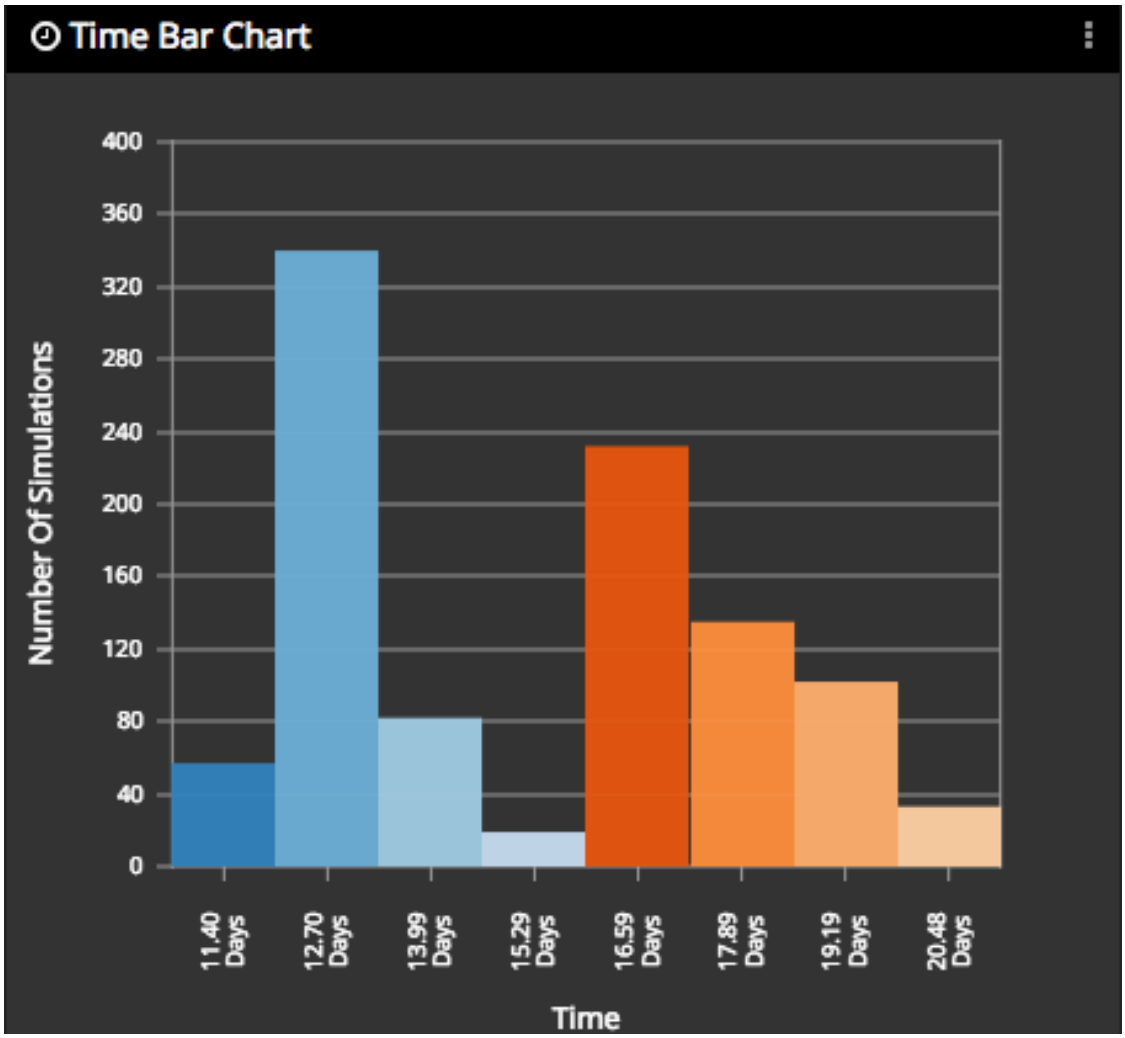



Figure 77. Monte Carlo Time Simulation with 1000 Trials for Perform Sprint Planning, Execution and Review (D.1.3.3)

For further insight, Figure 77 provides a bar chart showing the values returned for a 1000 iteration Monte Carlo simulation. The Y axis provides the number of simulations conducted for D.1.3.3 and the X axis provides the duration in days. Over 1000 trials the average effort to perform sprint planning, execution and review is approximately 15.9 days with 12.7 days being the mode. Based on actual productivity, there is a 50% chance of the sprint taking longer than 15 days, and a 40% chance of finishing in fewer than 13 days (see Figure 77). Since sprints are time-bound, any work not complete at the end of the sprint is moved to the product backlog. While the simulation shows that sprints may exceed the

time allocated for each sprint, in practice, no work will continue past the specified timeframe for each sprint.

## E. RECEIVE SOFTWARE DELIVERY AND DOCUMENTATION (D.1.3.4)

To perform the Monte Carlo simulation for "receive software delivery and documentation (D.1.3.4)," we assigned values to each decomposed action within D.1.3.4 (see Figure 78, Figure 79, and Figure 80). As with the previous simulation, no values were assigned to the higher-level D.1.3.4 action. Within Figure 79 and Figure 80 if an action is decomposed, any values assigned are attributed at the lowest level of the decomposition. For example, the action "review delivery against acceptance criteria (D.1.3.4.1)" in Figure 79, is shown to be decomposed; therefore, values are assigned to actions within D.1.3.4.1 as opposed to its parent action D.1.3.4.1 (see Figure 80). Values assigned to decomposed actions in Figure 79 and Figure 80 are included in Appendix A.

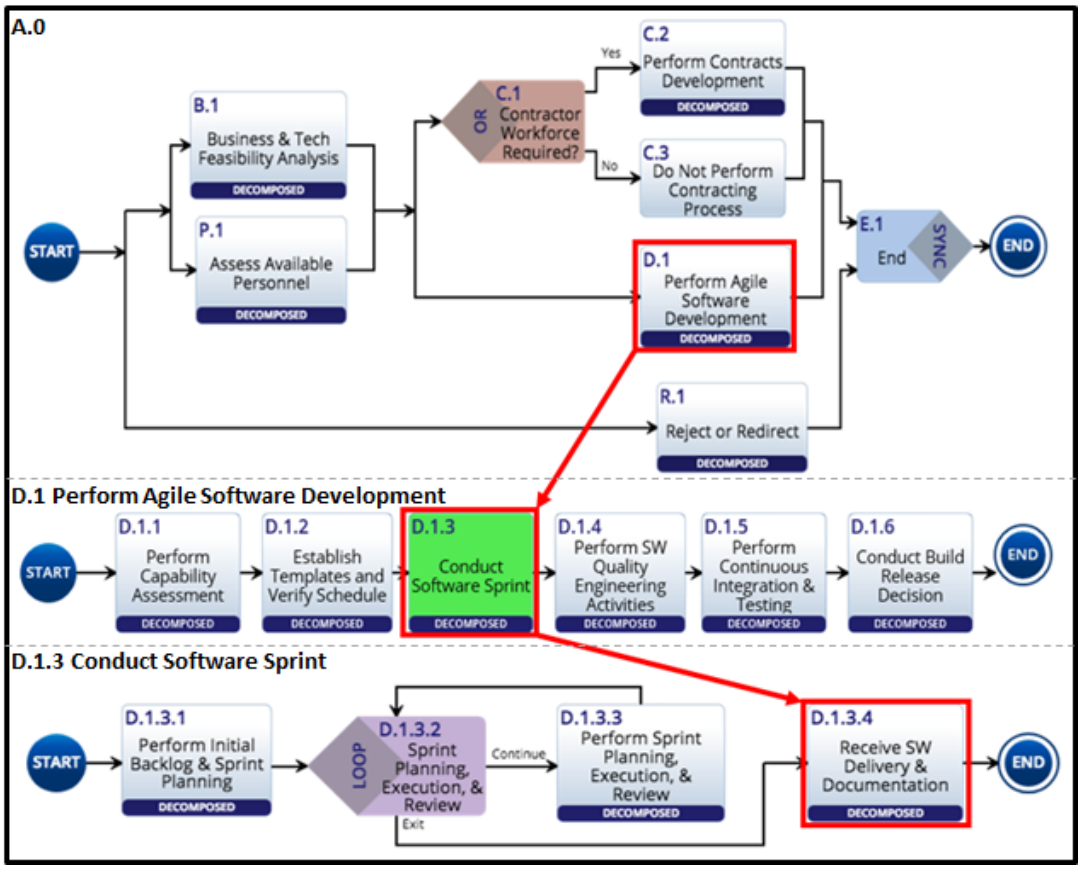



Figure 78. Decomposition Preceding Receive SW Delivery and Documentation (D.1.3.4)

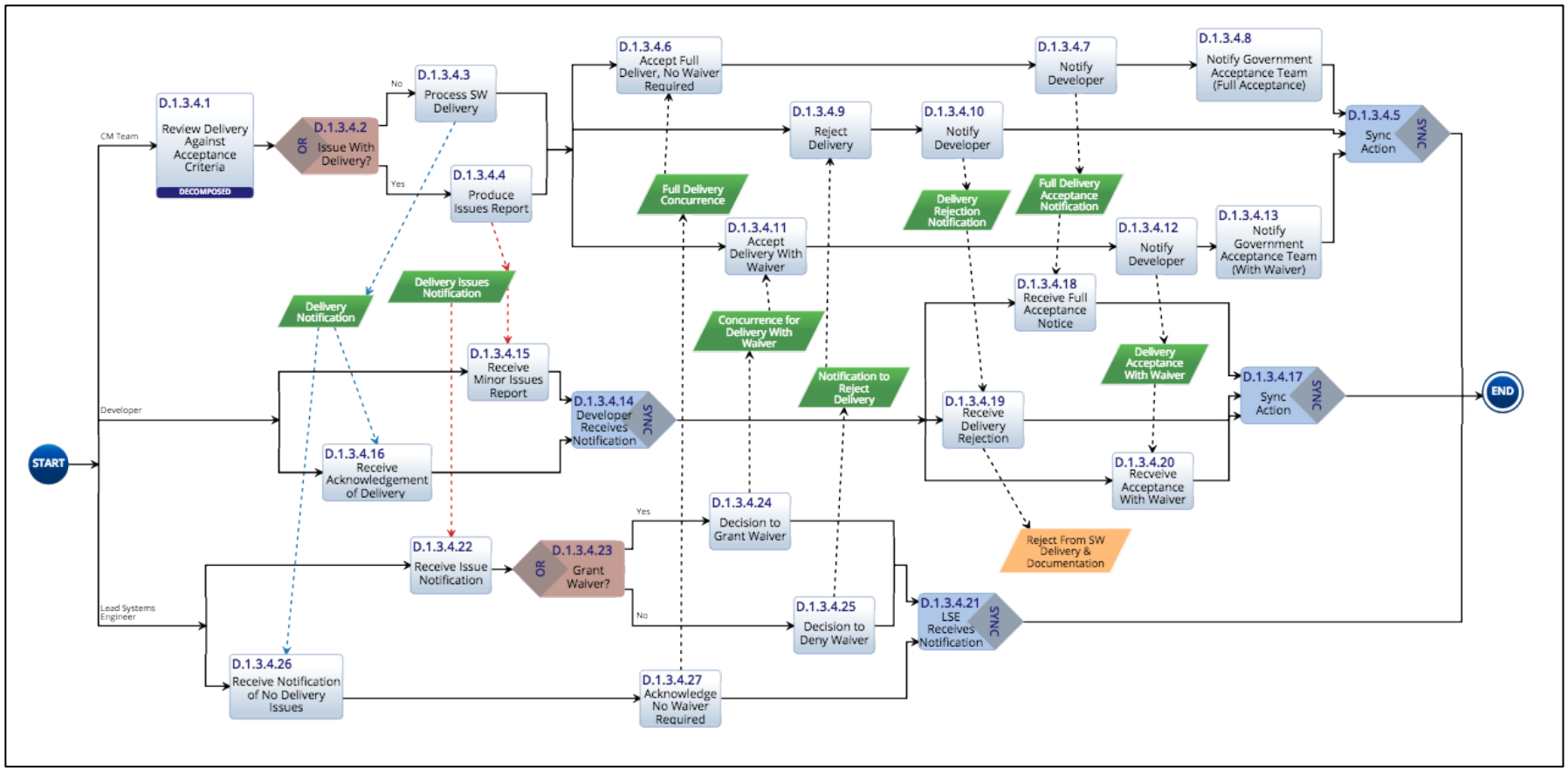



Figure 79. Decomposition of Receive Software Delivery and Documentation (D.1.3.4)

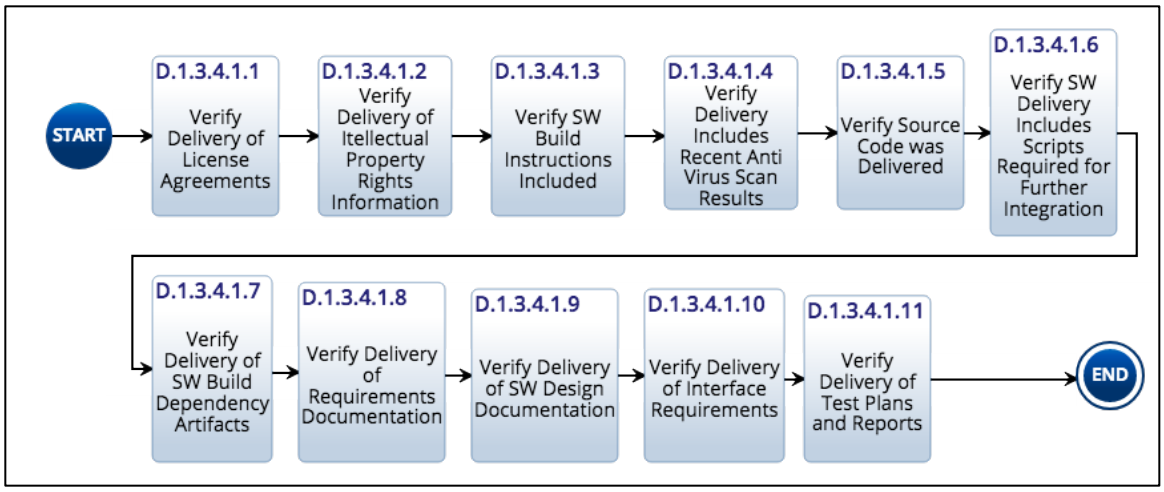



Figure 80. Decomposition of Review Delivery against Acceptance Criteria (D.1.3.4.1)

The results of the Monte Carlo simulation with 1000 trials for "receive software delivery and documentation (D.1.3.4)," are shown in Figure 81. Based on the simulation of D.1.3.4, the metrics show it will take an average of 5.3 days to perform D.1.3.4 with a standard deviation of 3.4 hours. The low standard deviation relative to the total average indicates increased fidelity in the results of the simulation. Additionally, it can be concluded that D.1.3.4 results in an average cost of $8,315. The total cost of D.1.3.4 is insignificant compared to the costs of D.1.3.1 and D.1.3.3. In addition, many of the actions performed, such as those within "review delivery against acceptance criteria" (D.1.3.4.1), are candidates for automation. Efforts to automate actions is expected to reduce effort and cost even further.

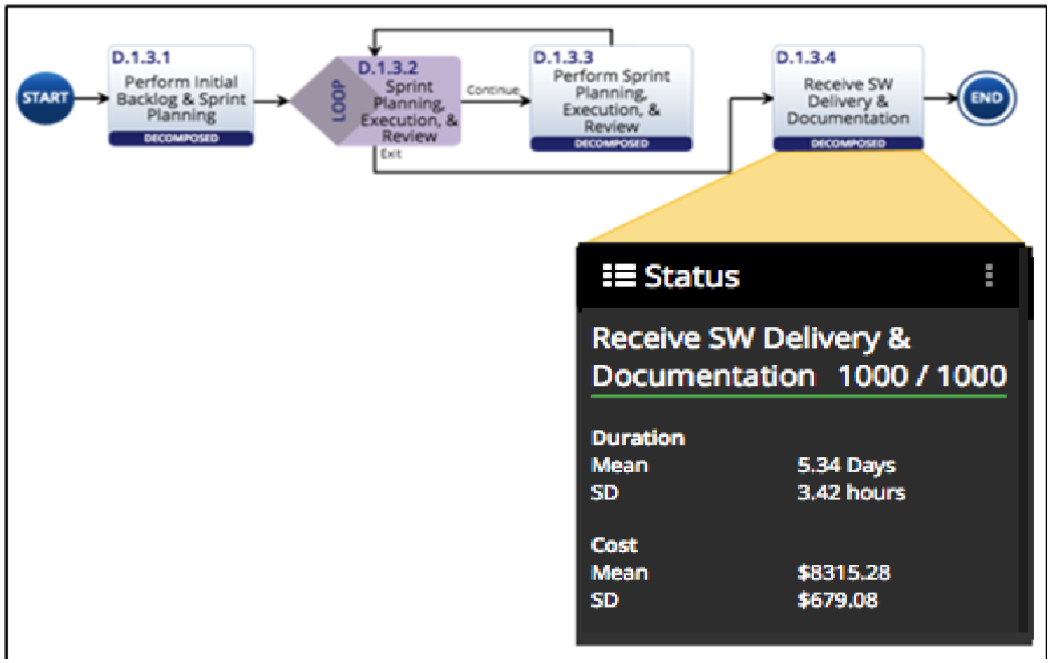



Figure 81. Monte Carlo Simulation Summary with 1000 Trials for Receive Software Delivery and Documentation (D.1.3.4)

A graphical depiction of the results for cost to perform "receive software delivery and documentation (D.1.3.4)," is shown in Figure 82. The bar chart shows the cost values returned for a 1000 trial Monte Carlo simulation, which has its Y axis provide the number of simulations conducted for D.1.3.4 and the X axis provide the cost in dollars. This chart shows two bars, which are representative of the two potential paths associated with the software delivery. The lower cost path reflects no issues with the software delivery whereas the higher cost path reflects issues with the delivery that required further analysis. The cost is projected to be either $7,187 or $8,538. The majority of this cost, whether the delivery is accepted or rejected stems from the action to "review delivery against acceptance criteria (D.1.3.4.1).

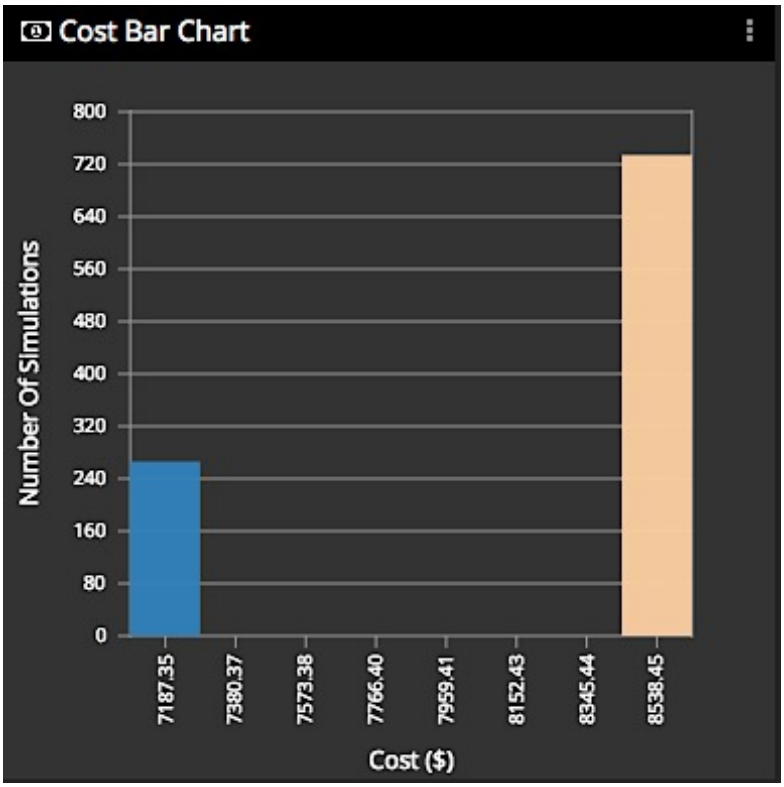



Figure 82. Monte Carlo Cost Simulation with 1000 Trials for Receive Software Delivery and Documentation (D.1.3.4)

For further insight regarding the amount of effort required to "receive software delivery and documentation (D.1.3.4)," Figure 83 provides a bar chart showing the values returned for a Monte Carlo simulation with 1000 trials. The Y axis provides the number of simulations conducted for D.1.3.3 and the X axis provides the duration in days. Effort is higher when issues with the delivery are present; however, the majority of time for this action is spent performing "review delivery against acceptance criteria" (D.1.3.4.1).

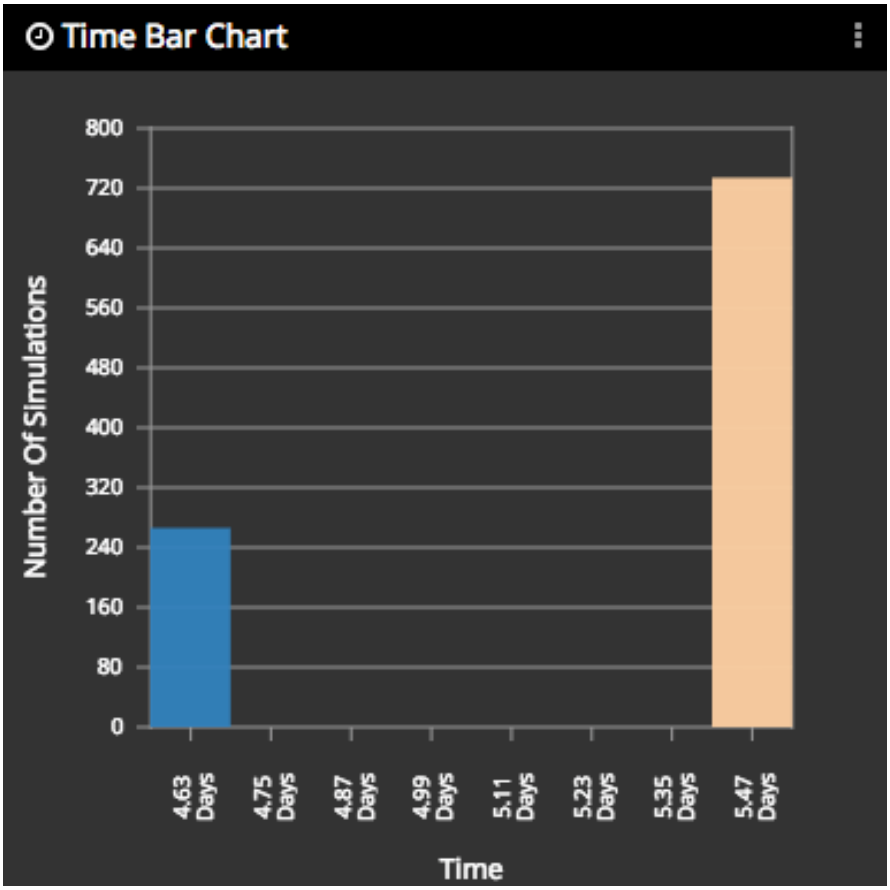



Figure 83. Monte Carlo Time Simulation with 1000 Trials for Receive Software Delivery and Documentation (D.1.3.4)

## F. SUMMARY SIMULATION OF CONDUCT SOFTWARE SPRINT (D.1.3)

Through review of "conduct software sprint (D.1.3)," Section F provides a holistic assessment of the Monte Carlo simulations for all three major corresponding supporting actions, which include "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)." The combination of these three groups of actions encompass the function provided by "conduct software sprint (D.1.3)." By combining the functionality of these actions in a single overarching action, aggregate metrics can be gathered based on simulation of the aggregate collection of activities for D.1.3 as shown in Figure 84.

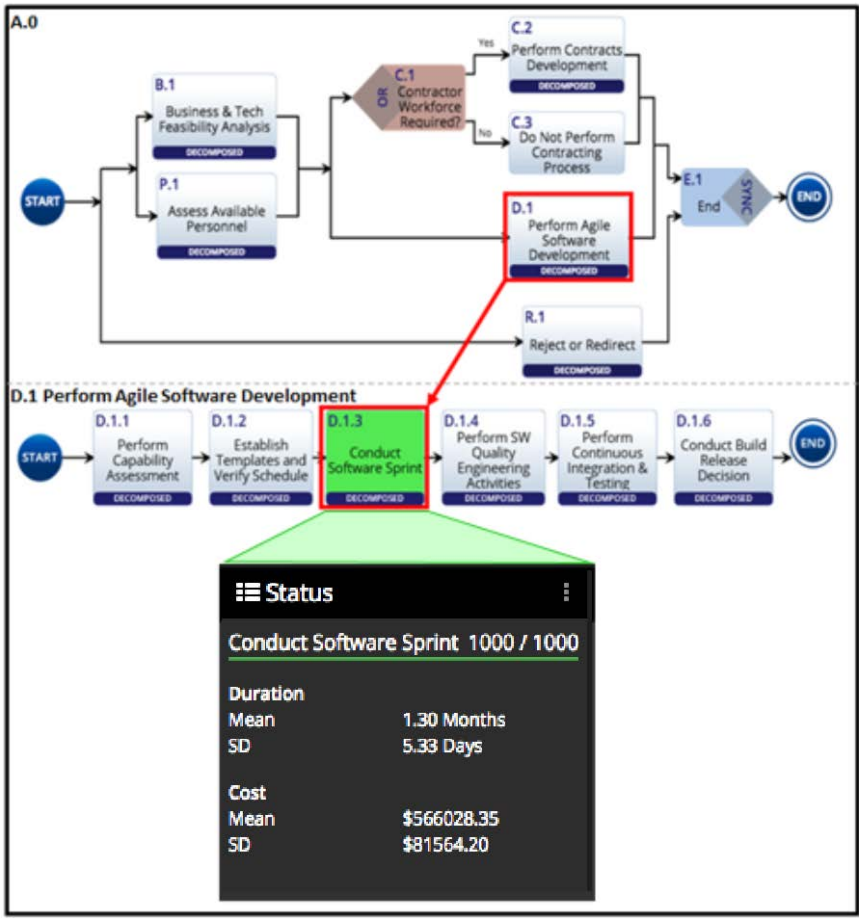



Figure 84. Monte Carlo Simulation Summary with 1000 Trials for Conduct Software Sprint (D.1.3)

The metrics shown in Figure 84 are driven by the costs built into the respective decomposed supporting actions "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)." Based on the outcome of the Monte Carlo simulation the action of "conduct software sprint (D.1.3)" costs $566,028 on average ($\mu$) with a standard deviation ($\sigma$) of $81,564. Although there is a lack of historical metrics for the number of people and hours for D.1.3.4, it did not have a significant impact on the overall simulation, since the action D.1.3.4 had a much lower cost impact than neighboring actions, D.1.3.1 and D.1.3.3. For perspective, D.1.3.4 costs $8,351 on average, whereas D.1.3.1 and D.1.3.3 cost an average of $304,395 and $254,434 respectively.

A bar chart of the overarching results for cost to “conduct software sprint (D.1.3)” is shown in Figure 85. The chart shows the cost values returned for a Monte Carlo simulation with 100 trials. The Y axis for the chart provides the number of simulations conducted for D.1.3 and the X axis provide the cost in dollars. This bar chart shows a normal distribution for 1000 iterations with a mean cost of $566,028. From review of the bar chart, we determined that approximately for 77% of the effort, cost is less than or equal to $557,655, which is slightly more than the average. This can be found by adding the number of simulations to the left of the bar for $557,655. Additionally, for 89% of the effort the cost ranges from $421,491 - $625,738. For a risk adverse manager, an estimate of $420K to $625,000 provides the lowest risk. If the range between the low and high point of that estimate range does not have enough fidelity, a manager could also evaluate that for 60% of the effort, cost will range between $489, 573 - $557, 655. This range is illustrated by the two highest bars in the chart.

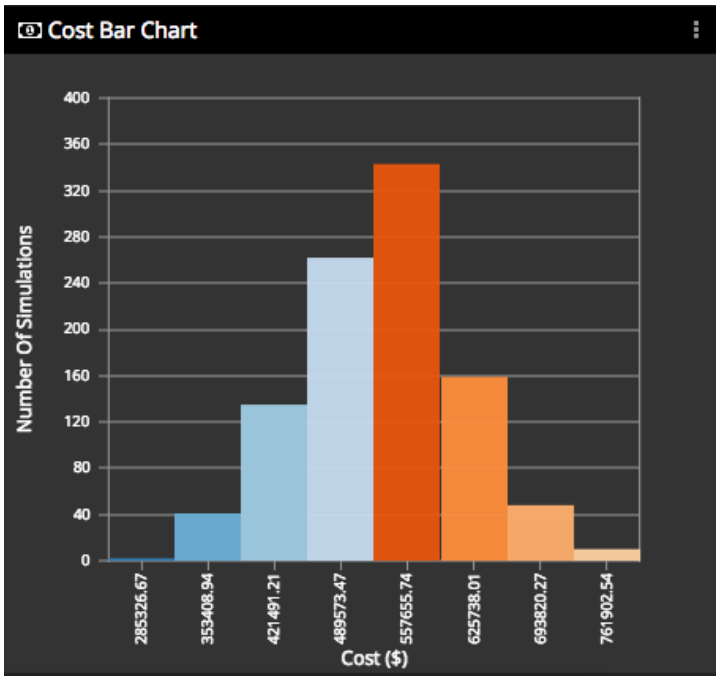



Figure 85. Monte Carlo Cost Simulation with 1000 Trials for Conduct Software Sprint (D.1.3)

The duration of “conduct software sprint (D.1.3),” is captured in the bar chart shown in Figure 86, which shows the values returned for a Monte Carlo simulation with 1000 trials. The Y axis provides the number of simulations conducted for D.1.3 and the X axis provides the effort in days and months. The chart supports the finding that the average effort of 1.3 months to conduct all activities in the software sprint. Adding the number of simulations in the time bar chart, there is a 74% likelihood that it will take between 1.07 - 1.34 months to complete all actions within D.1.3.

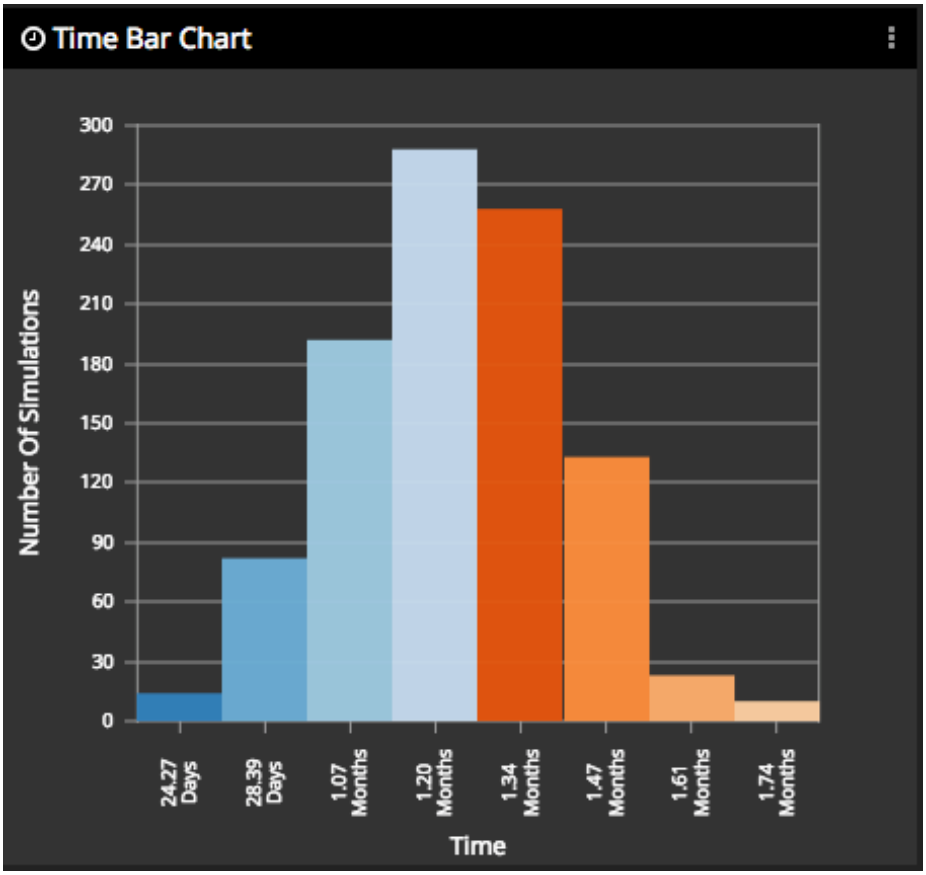



Figure 86. Monte Carlo Time Simulation with 1000 Trials for Conduct Software Sprint (D.1.3)

## G. MODEL USE CASE

This section provides a step-by-step example of how to modify the Innoslate model for "develop software code" (D.1.3.3.5.17). As presented in Chapter I, historical data with actual cost and effort was only available for the decomposed "develop software code" action. In addition to the availability of real-world data, the reason the action for "develop software code" is used for this use case is to provide a primer for model testing and validation discussion in Chapter V, which discusses model prediction measures for D.1.3.3.5.17 compared to planned and actual data.

With basic knowledge of Innoslate, the process architecture model can be modified to accept new input parameters, which can then be further analyzed. Innoslate provides the option to run either a discrete or Monte Carlo simulation. The effort required to update the model for different scenarios is dependent on the number of actions that need to be modified. For the model captured in this use case, the time to update the model can be measured in minutes for an Innoslate user familiar with the model. If the user has limited knowledge of the model, or there is a significant number of model parameters were required to be changed, the time to update could be substantially more, in terms of minutes, hours, and possibly days.

The use case scenario involves the following objectives:

- Change the number of sprint days from 10 to 15
- Change the number of software developers from 16 to 18
- Successfully run a Monte Carlo simulation and use the results

An initial simulation for a 10-day sprint with 16 developers results in a projected cost of $96,685 and 6.1 days (see Figure 87). The value of 6.1 days implies that the team of 10 software developers was productive for an average of 6.1 days out of 10. This value is based on assigning the average effort per developer of 4.8867 hours per sprint with a standard deviation of 0.6376 to "develop software code" D.1.3.3.5.17. The average and standard deviation are based on statistical analysis of project historical data (see Table 2). For this use case, we assume a project manager wants to know what the cost and effort

increase will be to go from 16 developers in a 10-day sprint to 18 developers in a 15-day sprint in order to plan for an increased workload. A summary comparison is provided at the end of this use case.


Figure 87. Use Case Initial Results

The following actions required to perform the use case:

- Step 1: Navigate to D.1.3.3.5.17 (see Figure 88).

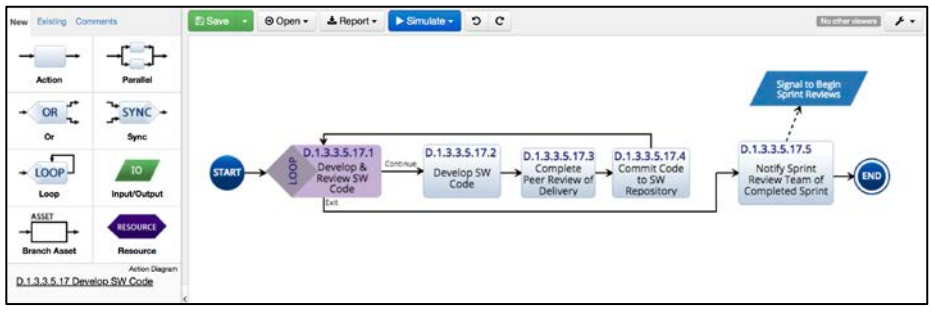



Figure 88. Use Case Step 1

- Step 2: Select the LOOP action for “develop and review software code,” D.1.3.3.5.17.1 (see Figure 89).
- Step 3: In the “attributes action” window, modify the mean and standard deviation values (see Figure 89).

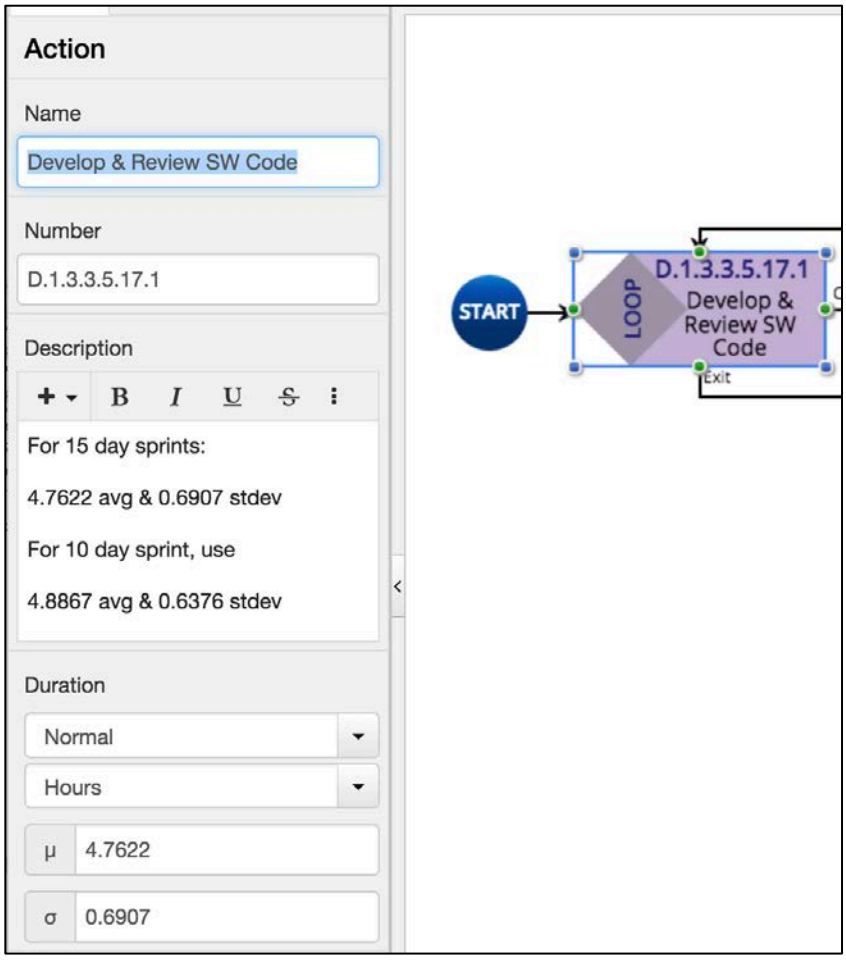



Figure 89. Use Case Step 2 and Step 3

- Step 4: In the menu bar, click “open,” then “entity view” (see Figure 90).

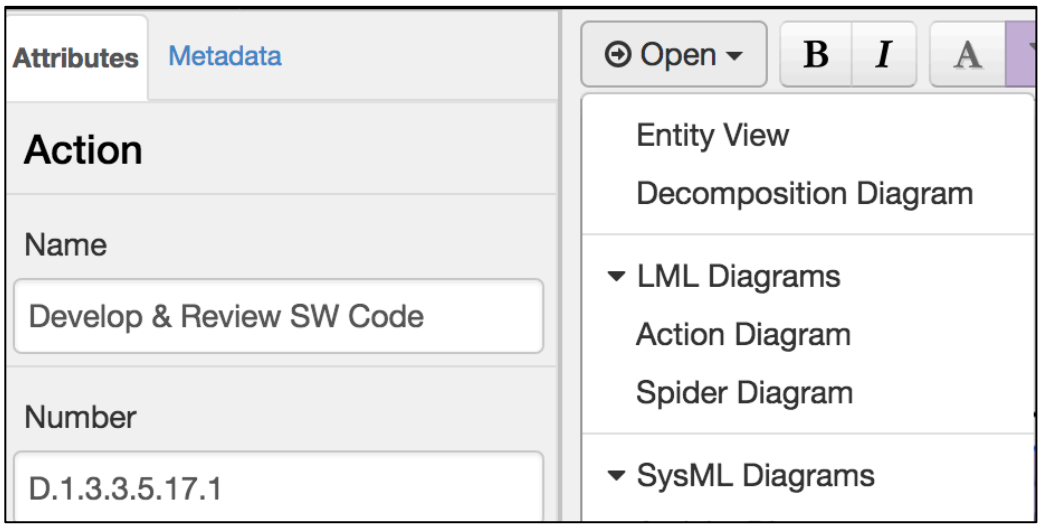



Figure 90. Use Case Step 4

- Step 5: Under “relationships,” click “program management” (see Figure 92).


Figure 91. Use Case Step 5

- Step 6: To increase the number of developers from 16 to 18, click “add,” “existing (default)” (see Figure 93).

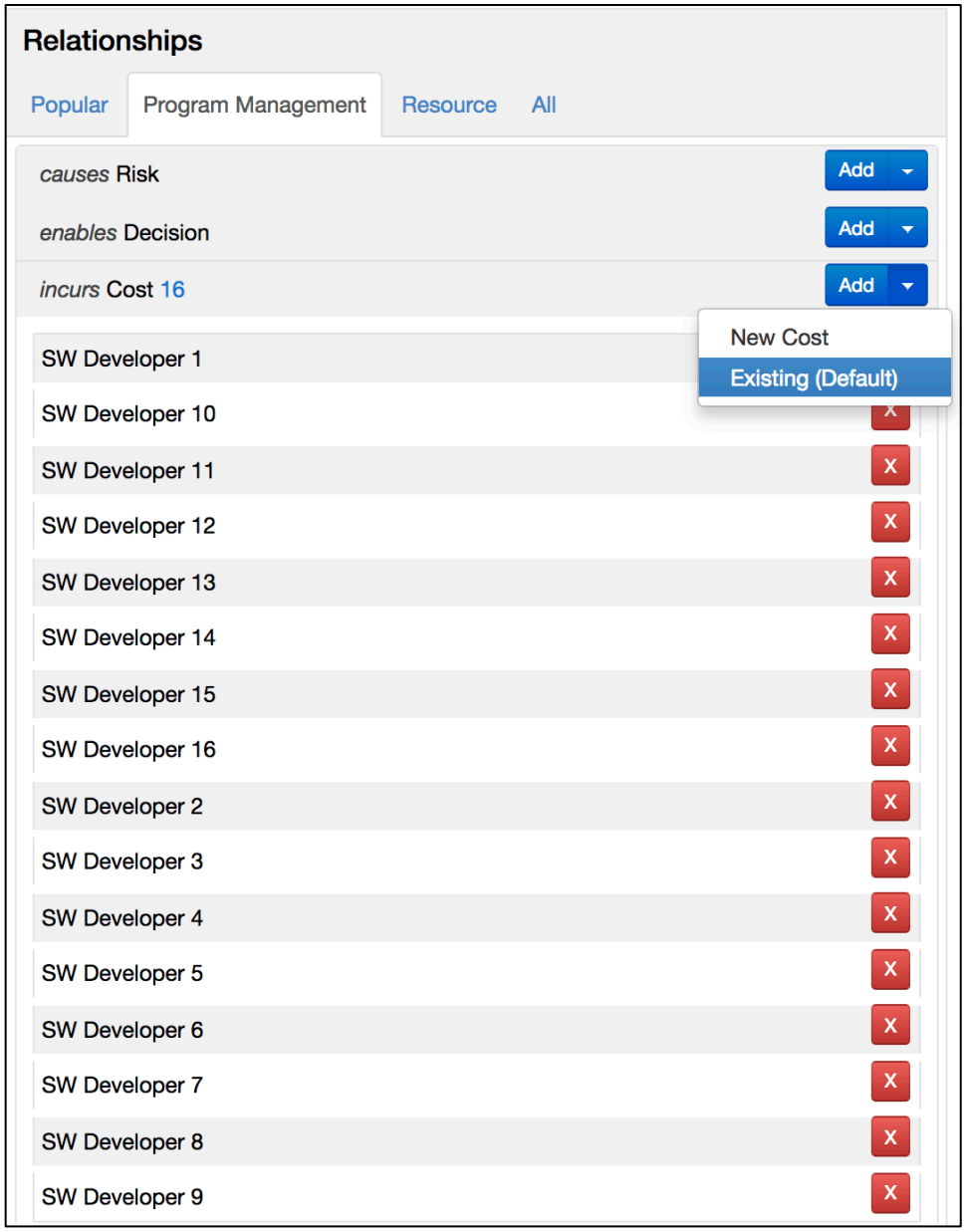



Figure 92. Use Case Step 6

- Step 7: Select the checkbox next to each software developer required to equal a total of 18 developers, then click “add” (see Figure 93).

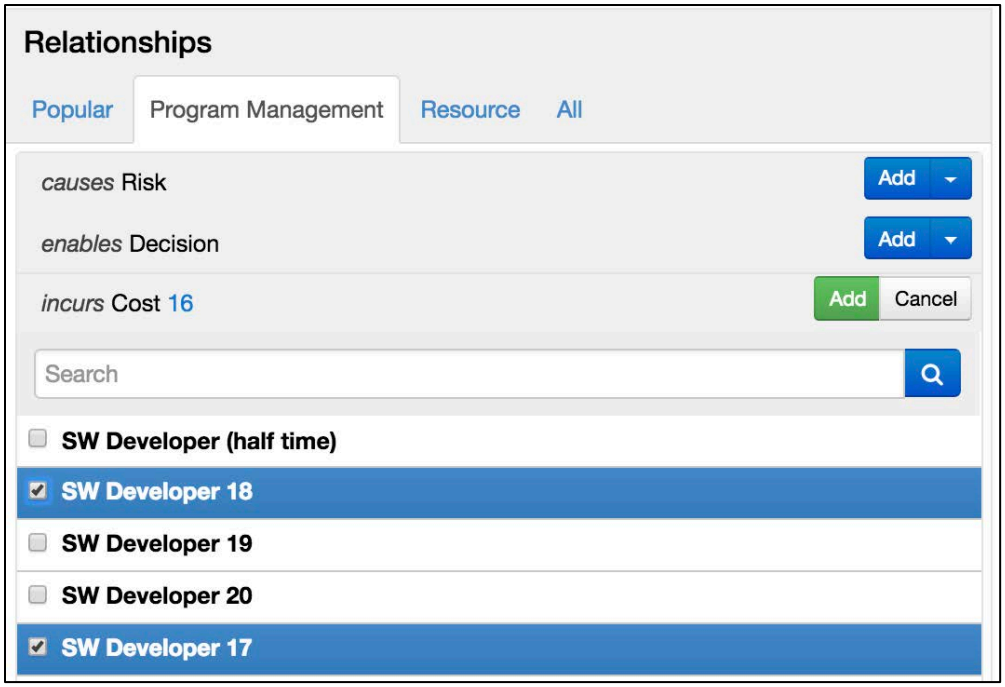



Figure 93. Use Case Step 7

- Step 8: Click “save” (see Figure 95).

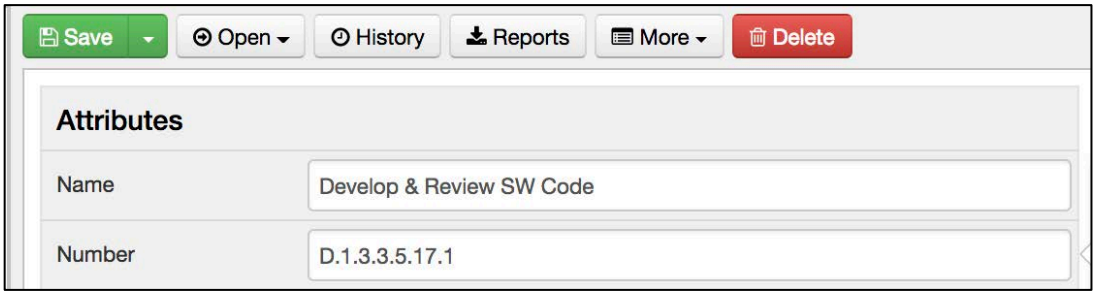



Figure 94. Use Case Step 8

- Step 9: Select the LOOP action for “develop and review software code,” D.1.3.3.5.17.1, then in the menu bar, click on “</> script” (see Figure 96).

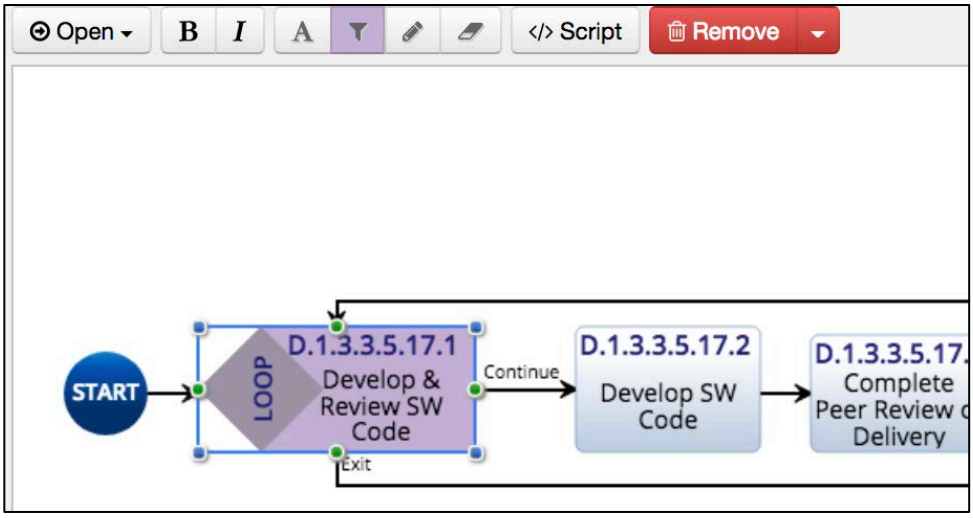



Figure 95. Use Case Step 9

- Step 10: In the “edit develop and review software code’s script” window, change the number of loop iterations from “9” to “14,” and click “done” (see Figure 96). The LOOP action will run once, then 14 more times, resulting in 15 loops. This effectively changes the number of sprint days from 10 to 15.

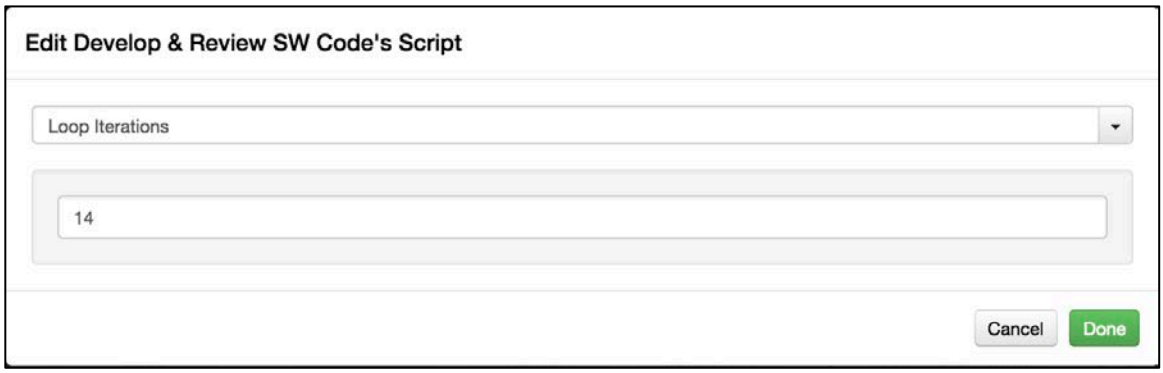



Figure 96. Use Case Step 10

- Step 11: Deselect the LOOP action for “develop and review software code,” D.1.3.3.5.17.1.
- Step 12: In the menu bar, click the “simulate” button, then click on “Monte Carlo” (see Figure 97).

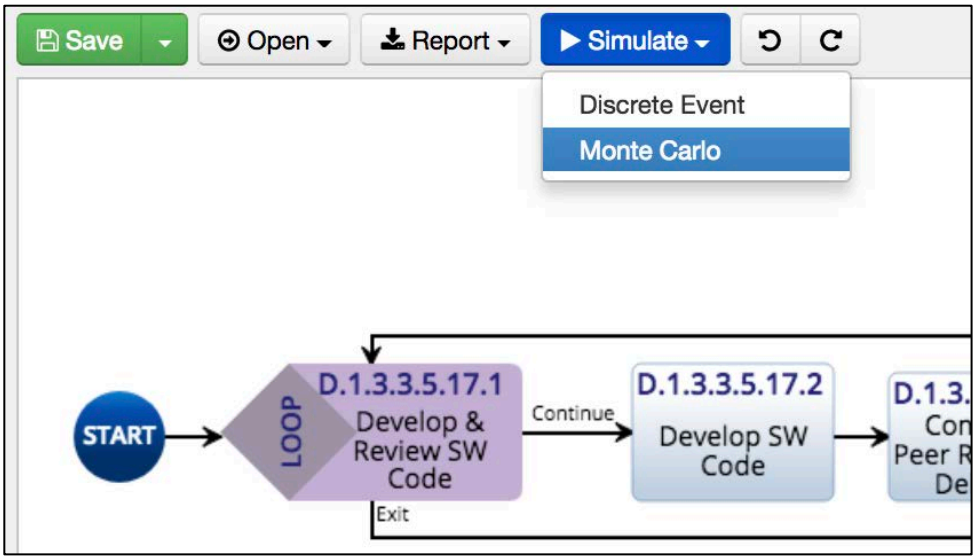



Figure 97. Use Case Step 11 and Step 12

- Step 13: Click on “settings” and ensure the “number of iterations” and “hours per year” values are correct for the intended simulation. For this model, 1000 iterations and 2,920 hours per year are used in order to generate simulation outputs for a standard eight-hour workday, found by multiplying 8 times 365 days in a year (see Figure 98).
- Step 14: Click “save settings” (see Figure 98).

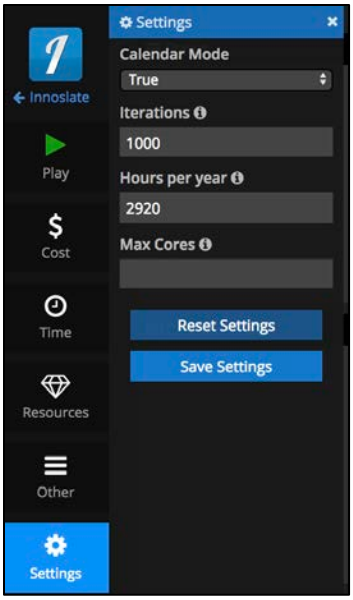



Figure 98. Use Case Step 13, 14, and 15. Source: SPEC Innovations (2017).

- Step 15: Click “play” (see Figure 98).
- Step 16: View output from simulation (see Figure 99).


Figure 99. Use Case Step 16

Analysis of changing the number of developers from 16 to 18 shows that effort increases from 6.1 days to 8.9 days and cost increases from $96,685 to $158,574. The simulation shows that there will be a cost increase of $61,889.

Table 3. Use Case Summary Comparison

| Number of Developers | Effort in Days | Cost |
|---|---|---|
| 16 | 6.1 | $96,685 |
| 18 | 8.9 | $158,574 |

## H. CHAPTER CONCLUSION

Chapter IV demonstrates model usage and Monte Carlo simulation by providing metrics to the action diagrams and architecture modeling described in Chapter III. The intention of the use case is to demonstrate how others could modify the model and use for different projects within the C2IS division. While the scope of the simulations is limited to data from a single software development project within SSC Pacific's C2IS division, the results have potential to be extensible to other agile software development projects within the organization utilizing the SOPs. Understanding the model constraints, how the values used for simulations were obtained, and the corresponding construct of actions provides context for understanding the output of the Monte Carlo simulation. The Monte Carlo simulation is completed at a high-level for the parent action of "conduct software sprint (D.1.3)" as well as individually for the dependent actions, which include "perform initial backlog and sprint planning (D.1.3.1)," "perform sprint planning, execution and review (D.1.3.3)," and "receive software delivery and documentation (D.1.3.4)." The simulations provide insight into the possibilities of using MBSE approaches to support cost and schedule estimates for agile software development within the C2IS division.

One benefit of having a model-based architecture is that it can be utilized as an experimental test bed for future projects. With the creation of an architecture that can simulate potential project parameters by using real-world data, there is a resulting necessity for projects to capture data for each major action to fuel the simulations. Gathering

additional real-world metrics will enable the model to be updated in order to perform further project analysis based on Monte Carlo simulations. The model use case provides an example of how model parameters can be modified for variables such as sprint length and personnel types assigned to various actions within the model in order to provide managers with cost and effort predictions Ultimately, having a well-defined and robust model as the architecture foundation provides a resource for refinement of holistic software development estimations and an aid for programs to quantifiably defend software development budgets.

# V. MODEL TESTING AND VALIDATION

## A. INTRODUCTION

Chapter V provides an analysis of model testing and validation to determine confidence in the model, and whether the model can accurately predict effort and cost.

> The modeler must establish his/her own confidence in the model and then convey that confidence to stakeholders and peers, usually through sharing results of Verification and Validation (V&V) exercises. Verification exercises determine whether or not a model is built correctly (error-free) and represents the intended behavior according to the model specification. Validation exercises determine whether the model provides an adequate representation of the real system for the model's stated purpose and addresses the sponsor's problem. (Madachy and Houston 2018, 26)

Model testing is scoped in terms of traditional post calibrated models and applicability of use in early phase cost estimating. In early phase cost models, "initial estimated inputs are the only information available for the early phase budgeting" and are used for model calibration (Rosa et al. 2017, 30). Traditional post calibrated models use final actual values for calibration and input parameters. Section B discusses model prediction accuracy using prediction measures such as relative error (RE), magnitude of relative error (MRE), mean magnitude relative error (MMRE), coefficient of determination ($R^2$), and PRED. PRED provides the percentage of cases in a data series that have an MRE value below a specified percentage, such as 20% for PRED(20) and 30% for PRED(30) (Boehm et al. 2000, 173). Section C provides a comparison of cost and effort estimation methods. Section D discusses threats to model validity. Section E presents chapter conclusions.

To establish model validity, we applied a series of verification and validation steps. While model prediction measures were limited to the cost estimates for software sprints, these verification and validation tests can be applied across the broader model to assess model validity. The process of model validation includes assessment 14 different structure and behavior tests covering "suitability for purpose, consistency with reality source, and utility and effectiveness of a suitable model" (Madachy 2008,119 - 121) (see Table 4, Table 5, and Table 6).

Table 4. Suitability for Purpose. Adapted from Madachy (2008).

| Focus | Test | Passing criteria | Results |
|---|---|---|---|
| Structure | Dimensional consistency | Variable dimensions agree with the computation using right units, ensuring that the model is properly balanced | Pass, all units consistent for person-hours, labor rates, and dollars |
| | Extreme conditions in equations | Model equations make sense using extreme values | Not tested: equations are built into tool, only input parameters are changed in the tool GUI (i.e., mean, STDEV, #loops, #developers) |
| | Boundary adequacy<br>-important variables<br>-policy levers | Model structure contains variables and feedback effects for purpose of study | Pass, model structure accounts for boundaries within the four SOPs |
| Behavior | Parameter (in)sensitivity<br><br>-behavior characteristics<br><br>-policy conclusions | Model behavior sensitive to reasonable variations in parameters<br><br>Policy conclusions sensitive to reasonable variations in parameters | Pass, model accepts changes in parameters<br><br>Policy considerations: Not tested: policy outside scope of research, see future work |
| | Structural (in)sensitivity<br><br>-behavior characteristics<br><br>-policy conclusions | Model behavior sensitive to reasonable alternative structures<br><br>Policy conclusions sensitive to reasonable alternative structures | Pass, model includes alternative structures based on user defined inputs such as number of developers and sprint duration<br><br>Policy considerations: Not tested: alternative software process policies outside scope of research, see future work |

Table 5. Consistency with Reality. Adapted from Madachy (2008).

| Focus | Test | Passing criteria | Results |
|---|---|---|---|
| Structure | Face validity<br><br>-rates and levels<br>-information feedback-delays | Model structure resembles real system to persons familiar with system | Pass, developers concur model matches factual SOP processes. 16/16 of developer role types and actions validated |
| | Parameter values<br><br>-conceptual fit<br>-numerical fit | Parameters recognizable in real system and values are consistent with best available information about real system | Pass with caveat for numerical fit: need larger sample size from more projects and more historical data |
| Behavior | Replication of reference modes (boundary adequacy for behavior)<br><br>-problem behavior<br>-past policies<br>-anticipated behavior | Model endogenously reproduces reference behavior modes that initially defined the study, including problematic behavior, observed responses to past policies and conceptually anticipated behavior | Pass, reproduces behavior modes within SOPS used to construct model<br><br>Past policies, model not suitable to simulate pre-agile process policies |
| | Surprise behavior | Model produces unexpected behavior under certain test conditions<br>1) model identifies possible behavior<br>2) model is incorrect and must be revised | Pass, discrete simulations revealed unexpected behavior when testing reject loops, and other action entities. Model revised to correct.<br><br>In addition, see future work related to Monterey Phoenix |
| | Extreme condition simulations | Model behaves well under extreme conditions or policies, showing that formulation is sensible | Did not experiment in this research, see future work |
| | Statistical tests<br>-time series analyses<br>-correlation and regression | Model output behaves statistically with real system data; shows same characteristics | Pass, model is statistically representative of real process. See details in Chapter V, section B for model prediction measures |

Table 6. Utility and Effectiveness of a Suitable Model.
Adapted from Madachy (2008).

| Focus | Test | Passing criteria | Results |
|---|---|---|---|
| Structure | Appropriateness of model characteristics for audience-size<br>-<br>simplicity/complexity<br>-aggregation/detail | Model simplicity, complexity and size is appropriate for audience | Pass, traceable to SOPs used to develop models for C2IS division |
| Behavior | Counter-intuitive behavior | Model exhibits seemingly counter-intuitive behavior in response to some policies, but is eventually seen as implication of real system structure | Pass, tested with varying levels of a standard day, 8hr and 24hr. Model exhibited counter-intuitive output until number of hours in year was modified to adjust to 8hr day.<br><br>In addition, see future work related to Monterey Phoenix |
| | Generation of insights | Model is capable of generating new insights about system | Pass, model is scalable, can be extended over time to address a larger process boundary, provides outputs to use for risk based cost and effort decisions |

## B. MODEL PREDICTION MEASURES

In order to test the ability of the simulation to predict effort and cost, we computed measures traditionally used for cost estimation models. The measures and criteria used to analyze the accuracy of the cost and effort simulation models include: error (E) (see Equation1), average RE (see Equation 2), MRE (see Equation 3), MMRE (see Equation 4), coefficient of determination ($R^2$) (see Equation 5), and the measure to assess the accuracy of model prediction (PRED) (see Equation 6 and Equation 7). The error (E) is calculated

by taking the difference between the estimated and the actual value for cost and person-hours worked per sprint (see Equation1). The RE as a measure of accuracy is calculated by dividing the error by the actual value for person-hours worked per sprint (see Equation 2). The MRE is the absolute value of the RE (see Equation 3). The MMRE is the average of the MREs (see Equation 4). The MMRE is an "indicator of model accuracy. Low MMRE is an indication of high accuracy. MMRE is defined as the sample mean (μ) of the magnitude of relative error (MRE)" (Rosa et al. 2017, 34).

The coefficient of determination ($R^2$) (see Equation 5) value is calculated by squaring the value obtained for the correlation coefficient (r). The equation for the coefficient of determination (Equation 5) is derived from the correlation coefficient equation found in the works of Mun (Mun 2015, 88). "The correlation coefficient is a measure of the strength and direction of the relationship between two variables, and it can take on a value between -1.0 and +1…the higher the absolute value of the correlation coefficient, the stronger the relationship" (Mun 2015, 88). In equation 5, the coefficient of determination is calculated by substituting actual historic values for the variable x and predicted measured values for the variable y. While a comparison of the $R^2$ values shows a wide variance, $R^2$ is just a cursory measure. A better indicator of the model prediction accuracy for individual sprints is the error value relative to the actual historic values, which is evident in the RE, MMRE, and PRED values.

The PRED(20) and PRED(30) results provide the percentage of cases in a data series that have an MRE percentage value below 20% for PRED(20) and 30% for PRED(30) (see Equations 6 and 7). For PRED(20) and PRED(30) values, a higher percentage indicates better performance. Conventional software cost models, such as Constructive Cost Model II (COCOMO II), generally attain a PRED(30) no better than 70%. In cases where the project is of shorter duration, PRED(20) provides greater accuracy can be attained for predictions; therefore, PRED(20) was used due to the short timespan associated with historical actual project data (see Table 7, Table 8, and Table 9). Considering the number of data points used in the simulation, it can be inferred that shorter duration projects, such as those that span weeks, will have higher accuracy estimates than projects spanning longer time periods, such as those that span multiple years.

$$E = MeasuredValue - ActualValue \quad \textit{Equation 1}$$

$$RE = \frac{E}{ActualValue} \quad \textit{Equation 2}$$

$$MRE = |RE| \quad \textit{Equation 3}$$

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \quad \textit{Equation 4}$$

$$R^2 = \left( \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - \left(\sum x_i\right)^2}\sqrt{n\sum y_i^2 - \left(\sum y_i\right)^2}} \right)^2 \quad \textit{Equation 5}$$

$$PRED(20) = \frac{1}{n}\sum_{i=1}^{n} x_i < 20\% \quad \textit{Equation 6}$$

$$PRED(30) = \frac{1}{n}\sum_{i=1}^{n} x_i < 30\% \quad \textit{Equation 7}$$

The “develop software code” (D.1.3.3.5.17) submodel provides the ability to perform traditional post calibrated analysis with a high degree of accuracy. The submodel accounts for the portion of work that is to be estimated using actual historical data. It can also be used for early-phase cost modeling; although, early phase cost modeling was proven to have a lesser degree of accuracy, as expected. Traditional post calibrated models use final actual values of effort, size and cost factors for calibration and input parameters. Early phase cost models are also calibrated to final effort, but use early phase cost factors for calibration and model inputs. The initial estimated inputs are the only information available for early phase budgeting on projects (Rosa et al. 2017, 30). In the case of early phase cost estimating where historical data might not be available, some assumptions may be required. When used for early phase cost estimating, model inputs are identical but may not be well known. For example, the model can be modified to use a standard hourly rate for personnel types, or an estimated number of software developers per sprint. The use case

discussed in Chapter IV demonstrates how to adjust the model inputs for "incurs cost" to assign various numbers of personnel to a specific action within the model. This provides the ability to use the model for early phase cost estimating.

As a traditional post calibrated model, actual effort and cost data for software development sprints are used as inputs in the simulation of the action diagram for "develop software code" (D.1.3.3.5.17). To evaluate prediction results, the output of the Monte Carlo simulation is used to compare actual effort and cost data to simulated model outputs. Comparison of simulated versus actual outputs provides the ability to verify whether predicted outputs for effort and cost correlates positively with actuals. Data for the comparison of simulated versus actual effort uses the mean output from 1000 trial Monte Carlo simulation. While software development subject matter experts from the C2IS division verified that the model factually aligns to the SOPs, the simulation results validate whether the model is able to predict accurate effort and cost estimates.

Historical actual data from five 10-day planned sprints and five 15-day planned sprints is used to calculate the average hours per day per sprint, and the standard deviation for each set of sprint lengths (see Table 1). The "LOOP" action within "develop software code" (D.1.3.3.5.17) was then assigned an average and standard deviation value based on sprint duration in days, either 10 or 15 days (Table 7, Table 8, and Table 9). Setting model simulation loop iterations incorrectly can result in an inaccurate output for the simulation. For example, setting the loop iterations to 15 days results in inaccurate simulation output for sprints that were only 10 days.

## C. COMPARISON OF COST AND EFFORT ESTIMATION METHODS

The prediction accuracies of the "develop software code" (D.3.1.5.5.17) simulation output using actual and simulated hours for 10 sprints are shown in Table 7. These results are generated using the traditional post calibration method. The $R^2$ value of 68.5% indicates that the simulated output for sprint duration closely tracks to the actual hours associated with a given sprint. The average RE value indicates that the typical output from the simulation is within 3.5% of the actual hours on average. The MMRE demonstrates the mean magnitude of difference between the estimated and actual values, which indicates

that the simulated duration outputs are within 10.3% of the actual hours on average. Analysis of average RE and MMRE shows that the simulated output is closely aligned to the actual output. The PRED(20) and PRED(30) values indicate that 90% of MRE values are less than 20%, and 100% of the MRE values are less than 30%. This performance indicates that 90% of simulation predictions are within 20% of actual sprint values, and 100% of simulation predictions are within 30% of actual sprint values. A visualization of the simulated versus actual hours against a perfect prediction line can be observed in Figure 100. The gray line in the scatter plot depicts what a perfect prediction output would look like relative to the simulated output for effort in our traditional post-calibrated model. No significant outliers appear within the comparison of actual versus simulated person-hours.

Table 7. Traditional Post Calibrated Model Prediction Accuracies for Develop Software Code: Actual vs. Simulated Person-Hours

| Sprint Number | Allocated Sprint Days | Actual Person-Hours | Simulated Person-Hours | Error | Relative Error (RE) | Magnitude Relative Error (MRE) |
|---|---|---|---|---|---|---|
| 1 | 10 | 781 | 859.6 | 78.6 | 10% | 10% |
| 2 | 10 | 1104 | 980.8 | -123.2 | -11% | 11% |
| 3 | 10 | 1038 | 982.4 | -55.6 | -5% | 5% |
| 4 | 10 | 912.5 | 984.0 | 71.5 | 8% | 8% |
| 5 | 10 | 929 | 982.4 | 53.4 | 6% | 6% |
| 6 | 15 | 1430.5 | 1221.3 | -209.2 | -15% | 15% |
| 7 | 15 | 950 | 1145.6 | 195.6 | 21% | 21% |
| 8 | 15 | 1217 | 1180. | -36.9 | -3% | 3% |
| 9 | 15 | 1400 | 1576.9 | 176.9 | 13% | 13% |
| 10 | 15 | 1314 | 1469.4 | 155.4 | 12% | 12% |
| $R^2$ = | 68.5% | | | Average RE = | 3.5% | |
| PRED(20) = | 90.0% | | | MMRE = | 10.3% | |
| PRED(30) = | 100.0% | | | | | |

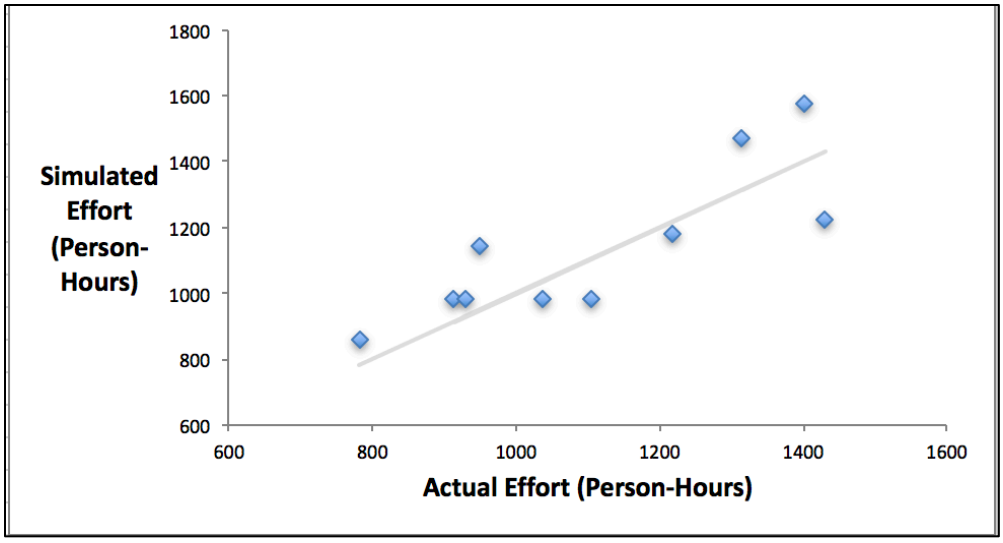



Figure 100. Software Development: Traditional Post Calibrated, Simulated vs. Actual Effort

The prediction accuracies of the "develop software code" (D.3.1.5.5.17) simulation output using actual and simulated cost for seven sprints are shown in Table 8. These results are also generated using the traditional post calibration method. The $R^2$ value of 59.3% indicates that the simulated output for sprint cost tracks relatively to the actual cost associated with a given sprint. The average RE value indicates that the typical output from the simulation is within 9.3% of the actual cost on average. The MMRE indicates that the simulated cost outputs are within 12.8% of the actual cost on average. Analysis of average RE and MMRE shows that the simulated output is well aligned to the actual output. The PRED(20) and PRED(30) values indicate that 85.7% of MRE values are less than 20%, and 100% of the MRE values are less than 30%. This indicates that 85.7% of simulation predictions are within 20% of actual sprint values, and 100% of simulation predictions are within 30% of actual sprint values. A visualization of the simulated versus actual cost against a perfect prediction line can be observed in Figure 101. The gray line in the scatter plot depicts what a perfect prediction output would look like relative to the simulated output for cost in our traditional post calibrated model. The perfect prediction line in the graph is based on actual values. Table 8 and Figure 101 show that are some outliers when comparing actual versus simulated cost. Future work may include further analysis of these outliers to ascertain their root cause.

Table 8. Traditional Post Calibrated Model Prediction Accuracies for Develop Software Code: Actual vs. Simulated Cost

| Sprint Number | Sprint Duration (in days) | Actual Cost | Simulated Cost | Error | Relative Error (RE) | Magnitude Relative Error (MRE) |
|---|---|---|---|---|---|---|
| 1 | 10 | $89, 257 | $103,327 | $19,057 | 21% | 21% |
| 2 | 10 | $126,171 | $120,578 | $640 | 1% | 1% |
| 3 | 10 | $118,628 | $120,751 | $7,877 | 7% | 7% |
| 4 | 10 | $104,285 | $120,965 | $22,114 | 21% | 21% |
| 5 | 10 | $106,228 | $120,853 | $20,294 | 19% | 19% |
| 6 | 15 | $163,485 | $150,441 | $(13,044) | -8% | 8% |
| 7 | 15 | $108,571 | $141,082 | $32,510 | 30% | 30% |
| $R^2$ = | 59.3% | | | Average RE = | 9.3% | |
| PRED(20) = | 85.7% | | | MMRE = | 12.8% | |
| PRED(30) = | 100.0% | | | | | |

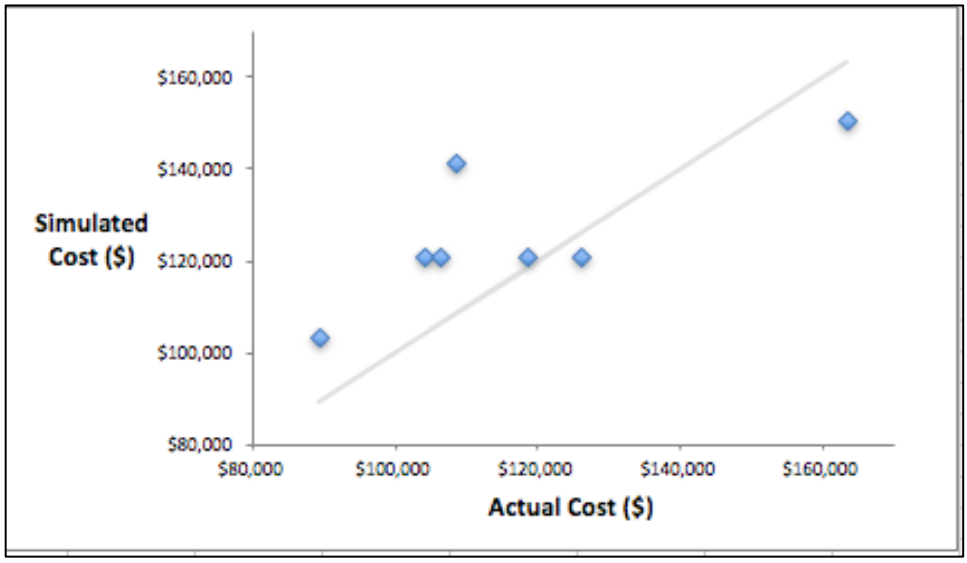



Figure 101. Software Development: Traditional Post Calibrated, Simulated vs. Actual Cost

The prediction accuracies of the "develop software code" (D.3.1.5.5.17) simulation output using actual and simulated cost for seven sprints are shown in Table 9. These results are generated using the early phase cost modeling approach via the use of a constant estimate of 18.5 developers for all seven sprints. The $R^2$ value of 31.4% indicates that the

simulated output for sprint cost tracks does not track relatively close to the actual cost associated with a given sprint. However, the low average RE value indicates that the typical output from the simulation is within 7.7% of the actual cost on average. The MMRE indicates that the simulated cost outputs are within 14.4% of the actual cost on average. Analysis of average RE and MMRE indicates that the difference between actual cost and simulated cost was not excessive across all sprints. The PRED(20) and PRED(30) values indicate that 71.4% of MRE values are less than 20%, and 85.7% of the MRE values are less than 30%. This indicates that 71.4% of simulation predictions are within 20% of actual sprint values, and 85.7% of simulation predictions are within 30% of actual sprint values. A visualization of the simulated versus actual cost against a perfect prediction line can be observed in Figure 102. The gray line in the scatter plot depicts what a perfect prediction output would look like relative to the simulated output for cost in our early phase model. The perfect prediction line in the graph is based on actual values. Table 9 and Figure 102 show that are some outliers when comparing actual versus simulated cost. Future work may include further analysis of these outliers to ascertain their root cause.

Table 9. Early Phase Model Prediction Accuracies for Develop Software Code: Actual vs. Simulated Cost

| Sprint Number | Sprint Duration (in days) | Actual Cost | Simulated Cost | Error | Relative Error (RE) | Magnitude Relative Error (MRE) |
|---|---|---|---|---|---|---|
| 1 | 10 | $89, 257 | $109,087 | $19,830 | 22% | 22% |
| 2 | 10 | $126,171 | $109,387 | $(16,784) | -13% | 13% |
| 3 | 10 | $118,628 | $109,228 | $(9,400) | -8% | 8% |
| 4 | 10 | $104,285 | $109,434 | $5,148 | 5% | 5% |
| 5 | 10 | $106,228 | $109,450 | $3,222 | 3% | 3% |
| 6 | 15 | $163,485 | $159,961 | $(3,524) | -2% | 2% |
| 7 | 15 | $108,571 | $159,870 | $51,298 | 47% | 47% |
| $R^2$ = | 31.4% | | | Average RE = | 7.7% | |
| PRED(20) = | 71.4% | | | MMRE = | 14.4% | |
| PRED(30) = | 85.7% | | | | | |

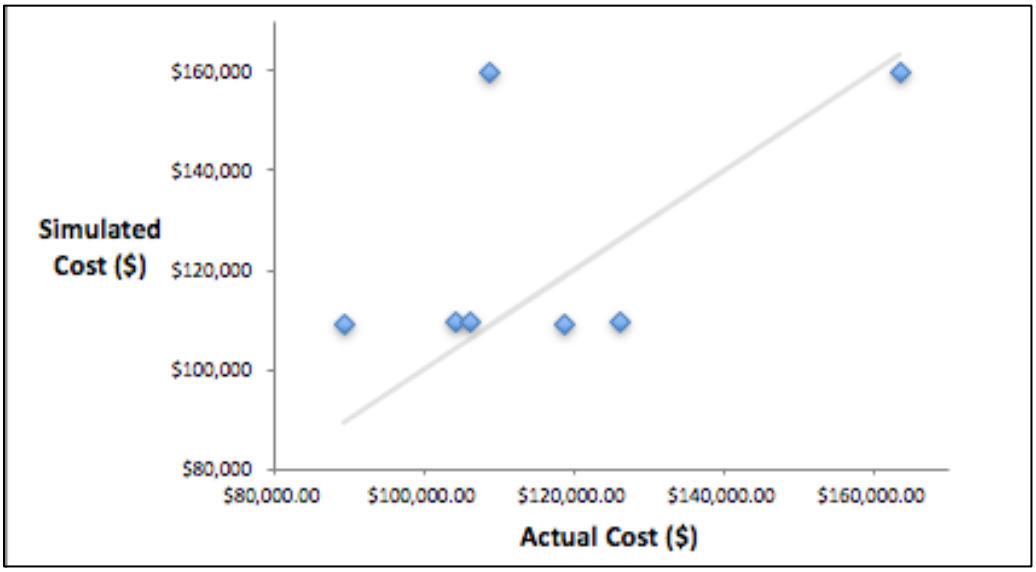



Figure 102. Software Development: Early Phase, Simulated vs. Actual Cost

Measures for actual versus planned cost evaluation criteria provides a benchmark for comparison with both traditional post calibrated and early phase prediction accuracies. For the planned cost estimation benchmark, the sprint planning team leverages subject matter expert knowledge and experience with past sprint efforts to develop an initial

planned estimate for the total number of person-hours and developers anticipated for each sprint based on the size, scope, and complexity of the work to be performed. A summary review of the evaluation criteria used to assess cost model prediction shows that the traditional post calibrated model provides greater prediction accuracy over the early phase cost mode (see Table 10).

While a comparison of the $R^2$ values shows a wide variance, $R^2$ is just a cursory measure; therefore, other measures such as MMRE and PRED must be evaluated as they are better indicators of model prediction accuracy. A comparison of MMRE and PRED(20) for planned cost estimates versus simulation using final actuals shows that the simulation reasonably predicts benchmarks. Although the model can be used for early phase cost modeling, analysis of the evaluation criteria shows that the model performs better when calibrated using historical actual data. The prediction accuracy of the model indicates that it can be used in lieu of the project team's current planning approach for cost and effort estimation.

Table 10. Comparison of Cost Estimation Methods

| Evaluation Criteria | Planned Cost Estimates | Simulation Using Final Actuals | Simulation Using Initial Estimates |
|---|---|---|---|
| $R^2$ | 91.9% | 59.3% | 31.4% |
| Average RE | 7.3% | 9.3% | 7.7% |
| MMRE | 7.7% | 12.8% | 14.4% |
| PRED(20) | 100% | 85.7% | 71.4% |
| PRED(30) | 100% | 100% | 85.7% |

## D. THREATS TO VALIDITY

While we were able to utilize historical actual project data to perform Monte Carlo simulations for software development sprints, and the model prediction results are reasonable, there are still threats to model validity. The actual metrics obtained are from a software project that is still in progress; therefore, the model prediction measures are only based on a small sample size. Without the whole project represented in the data set used to

conduct model prediction analysis, subsequent data collection and analysis may result in different average and standard deviation values assigned to various action entities within the model. Additionally, data fitting tools can be used with a larger sample size to ascertain if another distribution model, such as triangular, normal, or exponential provide a better fit. Also, there is no absolute evidence that the data collection for analysis is 100% accurate.

Effort in person-hours is captured by the project; however, this data capture is still subject to human error and bias when developers record the number of hours they worked into the tools used to track hours. Model validity is also currently tested with historical actual data used to calibrate the model. Extending model validity to test with data not calibrated to the model may yield different results. Examples include using data from a non-DoD project to test model validity, or separating calibration and validation data. It is also possible that the model cannot be generalized to a non-DoD project. Although the process architecture was built from SOPs that were created using academic and industry best practices, bias from SOP authors could manifest in intentional or unintentional bias within the SOPs and corresponding process architecture. Threats to validity may also include discrimination in data selected for analysis; therefore, modelers must avoid only picking data that would make the model outcome favorable.

## E. CHAPTER CONCLUSIONS

The process architecture model effectively emulates software development processes used in the organization based on SOPs, and as such, it can be used to perform cost and effort estimations in lieu of the project team's current planning approach for cost and effort estimation. The model passes structure and behavior validation and verification tests for "suitability for purpose, consistency with reality source, and utility and effectiveness of a suitable model" (Madachy 2008,119–121). Evaluation criteria used to compare actual versus simulated effort and cost, and actual versus simulated cost showed that the model's prediction results are favorable. While model performance is greater when used as a traditional post calibrated mode, model prediction measures also indicate that the model has potential to be used as an early phase cost model.

Regarding cost estimation, the MMRE value for simulation using final actuals is 12.8% and 7.7% for planned cost estimates. The corresponding simulation PRED(20) of 85.7% indicates that, on average, the validation results generate estimates that are within 20% of the actuals, 85.7% of the time. Planned cost estimation values have a PRED(20) value of 100%. A comparison of MMRE and PRED(20) values shows that the simulation performed reasonably well when compared to planned and actual values. We conclude that our model has a high degree of prediction accuracy relative to actual values.

With respect to effort estimation, the MMRE value for simulation using final actuals is 10.3% and 11.6% for planned effort estimates. The corresponding simulation PRED(20) of 90% indicates that, on average, the validation results generate estimates that are within 20% of the actuals, 90% of the time. Planned effort estimation values have a PRED(20) value of 80%. A comparison of MMRE and PRED(20) values shows that the simulation performed reasonably well when compared to planned and actual values. Since both MMRE and PRED(20) simulation results are more accurate than planned values, we conclude that our model has a statistically significant degree of prediction accuracy relative to actual values.

THIS PAGE INTENTIONALLY LEFT BLANK

# VI. FINAL CONCLUSIONS

## A. FINDINGS AND RESULTS

To achieve the goal of providing elegantly engineered command and control capabilities for naval, joint, and national level customers, a well thought out integrated process architecture was developed. The action and IDEF0 architecture diagrams generated for this project provide the C2IS division with a holistic architecture that yields an integrated view of the current stove-piped SOPs. This architecture demonstrates and identifies critical interdependencies of the current stand-alone SOPs while providing an easily understood flow of interconnected activities defined within the SOPs. Since the architecture for the SOPs has an understandable flow, it has potential to assist a new project manager in facilitating the execution of an agile software development cycle within the C2IS division. Additionally, the end product architecture can be used to assist with cost estimation simulation of software development processes to bypass the need to expend actual resources in learning by trial and error. Given the prospective benefits of this architecture and simulation-based action diagrams, the results of this project will be used for potential adoption by other software projects within the C2IS division.

Five key takeaways were gleaned from the creation of a holistic MBSE agile software development architecture. First, by following an LML approach for building the SOP architecture, the development of corresponding IDEF0 diagrams followed a natural progression that ensured a thorough capture of the requisite SOP functions. Creating action diagrams first is a tenet and best practice of an LML approach for building corresponding IDEF0 diagrams; in practice, this enabled the development of integrated functions.

Second, using LML action diagrams, an integrated architecture was developed that encompassed the four core SOPs. Architecture form and function were captured in detail by the multi-level decomposition of the six high-level functions within the SOP architecture. The integrated architecture with proper form and function mapping provides a means for addressing the issues with stove-piped SOPs.

Third, the core SOPs adequately captured the processes for business, contracting, and personnel activities to develop the LML representations of action diagrams and functional architecture. Utilizing a holistic architecture enables a more comprehensive assessment of the SOP actions and functions. A significant finding from this model development is that the current SOPs adequately cover contracting for software development, pre-vetting and capability assessments of S&T based software, and processes for continuous integration and testing. However, none of the current SOPs that we analyzed provide guidance for how to perform agile software design and development actions followed by existing projects within the organization. Based on this finding we developed LML action diagrams that captured the process for performing agile software development. These processes were verified by software development SMEs within the C2IS division, and subsequently validated through simulation.

Fourth, successful application of metrics to simulate software development sprints within the architecture produced a model that accurately reflects the agile software development environment within the C2IS division. To validate the architecture against reality, real-world metrics were obtained from a C2IS division project to conduct simulations within Innoslate. Viable metrics were acquired, but in the process of collecting metrics, we discovered that metrics did not exist for all modeled tasks. Metrics were available for the planned and actual number of developers involved in a software development sprint; however, the project was not tracking the planning work leading up the sprint and the review work after the sprint. The model that we developed established the framework for project personnel to begin collecting metrics within the verified and validated model for future use by other software development projects within the C2IS division.

Fifth, by running simulations for the software sprints, it is possible to obtain realistic outputs from the model that can facilitate analysis of effort and cost required to conduct software sprints within the division. The model was validated using a battery of validation tests, and outputs were tested with statistical analysis against historical data. The process architecture model produced simulated outputs that were comparable to historical effort and cost data from the C2IS division software project. By creating a model that

accurately represents real-world software development processes, model parameters can be modified to optimize simulation output, and, subsequently, optimized parameters can be applied to corresponding real-world processes to reduce effort and cost. One benefit of having a model-based architecture is that it can be utilized as an experimental test bed for future projects.

## B. FUTURE WORK AND RESEARCH

Given the limitations to the scope of the analysis performed for this project, there are certain aspects that would benefit from additional research. The holistic agile architecture does not currently include integrated diagrams for the deployment, maintenance, or retirement of software post-development. Additional research can be performed to further assess these aspects. The current model does not prescribe the physical mechanisms for how the functions are executed. For example, if cloud-based technologies are used as the mechanism to develop and deploy software, then the architecture could be updated for cloud-based and automated technologies. With respect to quantifiable data for Monte Carlo simulations, additional metrics can be obtained for business, contracts, and personnel functions within the C2IS division to expand the scope of simulations.

The model accounts for process flows as-is within the C2IS division. Future work will be required to ensure the model reflects the latest software development methodology used by the C2IS division. In addition, the sample size of data used was only from a single project for 10 sprints with a duration of 10 or 15 days. There are additional software development projects within the C2IS division that are able to provide data for future use in this model. To increase validity, data from additional projects can be included and analyzed to update model parameters. While the model is currently best suited as a traditional post calibrated model, future work can continue to improve the model in order to explore its possible use as an early phase cost model. For example, stronger tests of prediction can be performed by segmenting the calibration and validation data, then attempting to predict sprints that are not included in the calibration. With larger samples, local data can be segmented in order to assess the prediction in another environment where no data was collected (Boehm et al. 2000, 173).

The model only accounts for the number of hours and cost per personnel role. The model does not account for complexity or quality of the software sprints. For example, the actual effort of tasks planned and completed may vary widely from sprint to sprint and project to project based on the complexity of tasks allocated to sprints. This could be a reason for the outliers shown in the model prediction accuracy tables, but future work is required to determine the root cause of these outliers. Future work could also include updating the model to account for complexity and quality of software development. The model for "conduct software sprint," D.1.3, uses the same hourly rate for all software developers. In real-world operations, the software development team will be comprised of a mix of junior, mid-level, and senior developers. Future iterations of the model should account for the various levels of developer seniority or experience rather than using a single rate for all developers.

Due to the potential for changes in business, contracts, personnel, or software development processes, the structure of the SOPs may change. Consequently, future work could include updating the models to reflect changes in the SOPs. Lastly, there is no current SOP for how to perform agile software development within the C2IS division. To address this gap, an MBSE process architecture could be developed for use as or in conjunction with newly developed SOPs.

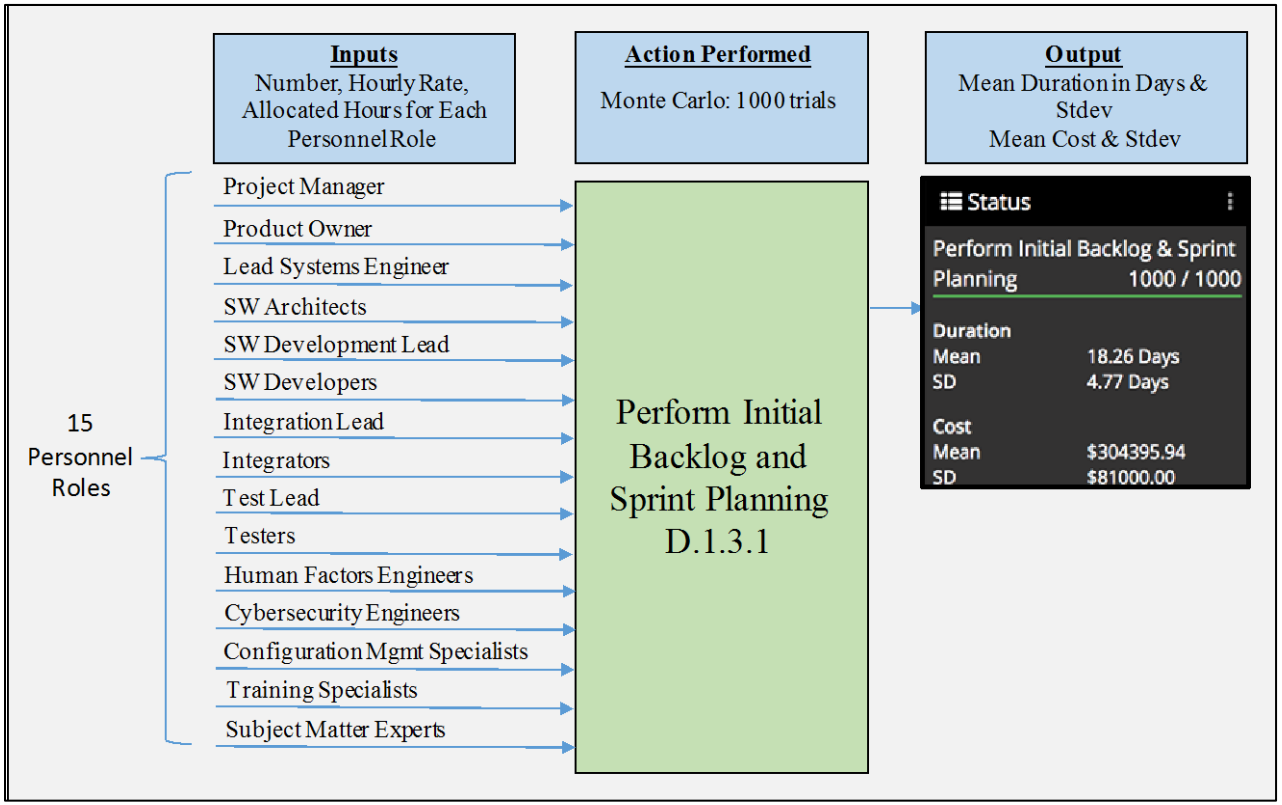
Future work that is related to existing research includes further exploration and comparison of the research by White (2014) to provide greater insight into the ability to quantify rework as part of the process architecture. De Silva, Rayadurgam, and Heimdahl (2015) focused on agent-based modeling and the decoupling of product and process. While product and process were integrated within our process architecture, further study to determine the impact of decoupling product and process within the model is warranted. Glaiel, Moulton, and Madnick (2013) defined a set of core characteristics that are the essence of agile, two of which were continuous integration and customer involvement. While our process architecture includes customer interaction as a set of actions that occur, it does not fully decompose how the customer interacts within the development process. Agent-based modeling could be valuable in determining how customer interaction can be modeled or even optimized as part of the agile development process. Additionally, research

by Cao, Ramesh, and Abdel-Hamid (2010) explored the impact of requirements volatility using system dynamics to account for interdependencies within the agile development process. Future work may include exploration of requirements volatility and how it can be accounted for within our process architecture. The research by Moutlon et al. (2017) explored the scaled agile framework (SAFe). Future work to modify and update our process architecture to scale for an enterprise level application. Giammarco's (2012) AMBIA dissertation provided insight into heuristics for architecture development. Future work may help determine if these heuristics are also applicable to software process architectures in order to help bridge the gap between traditional systems engineering and software development. Lastly, Rosa et al. (2017) focused on early phase cost modeling with DoD environments. While data collection, measures of validity, and staff size is accounted for in our process architecture, product size or complexity were not quantified within the model and is an area for further exploration and collaboration.

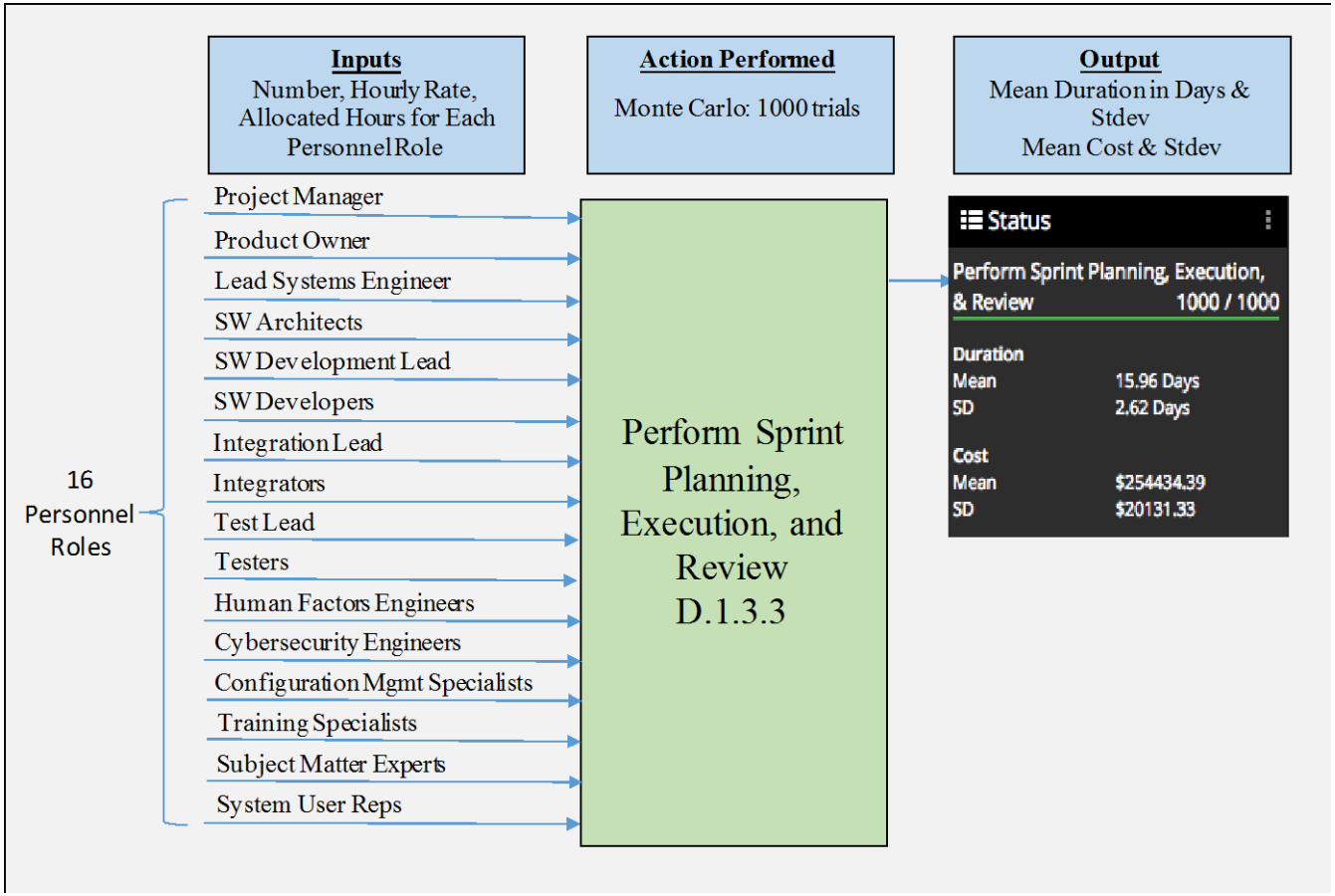
Besides updating the process architecture model, other approaches may be pursued as well. Analysis and development may be repeated using Monterey Phoenix to gain an understanding of how its exhaustive scenario generation approach compares with the Monte Carlo simulation approach in terms of accuracy for effort and cost. Monterey Phoenix can also be used to supplement the process architecture by exposing model surprise behavior for further verification and validation. This helps determine how consistent the model is with reality, specifically whether or not the model produces unexpected behaviors.

THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX A. PROJECT DATA COLLECTED

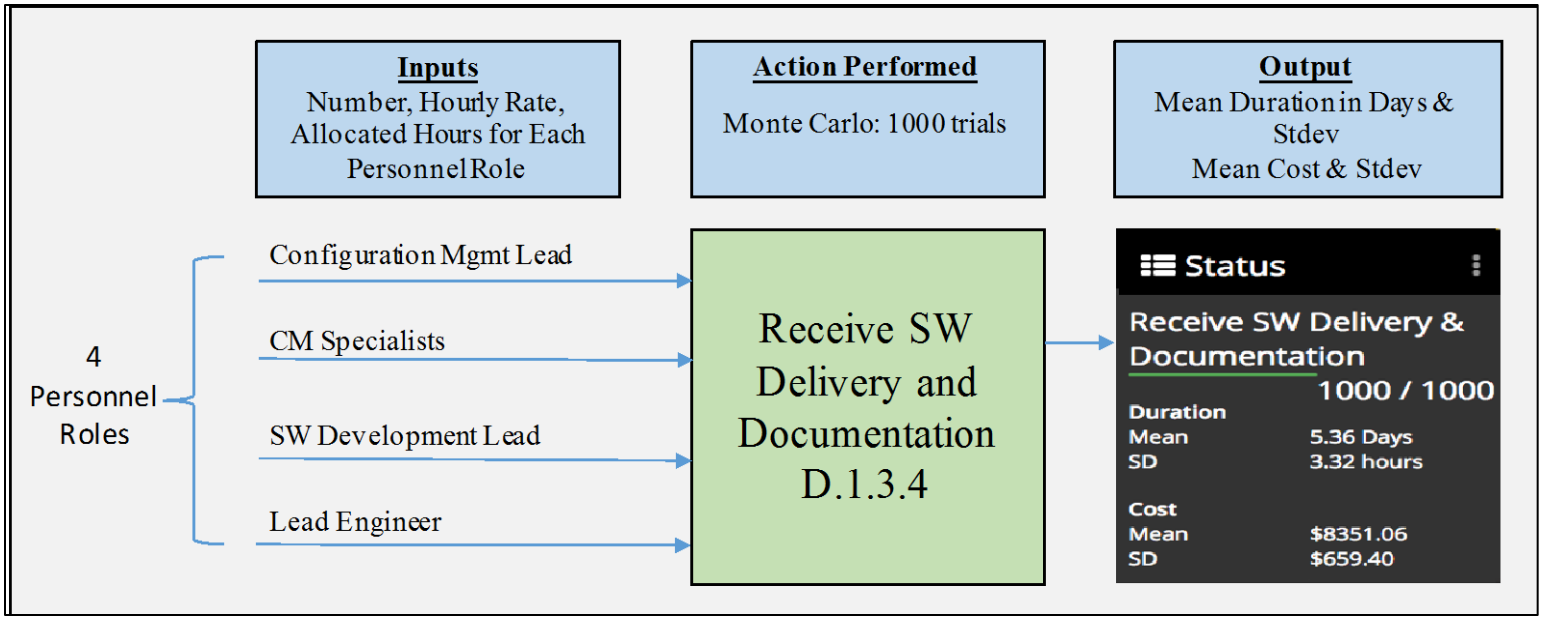


| | | | Perform Initial Backlog and Sprint Planning (D.1.3.1) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Define Roles and Responsibilities (D.1.3.1.1) | | | Initialize Product Backlog (D.1.3.1.3) | | | Build Review & Release Planning (D.1.3.1.2) | | |
| Role | GS-Equiv | Hourly Rate (FY18 burdened) | # of FTEs | Allocated per FTE | Cost Estimate Using Avg | # of FTEs | Hours Allocated per FTE | Cost Estimate Using Avg | # of FTEs | Hours Allocated per FTE | Cost Estimate Using Avg |
| Project Manager | GS-15 | $ 154.14 | 1 | 8 | $ 1,233.12 | 1 | 16 | $ 2,466.24 | 1 | 4 | $ 616.56 |
| Product Owner | GS-13 | $ 126.50 | | | $ - | 1 | 160 | $ 20,240.00 | 1 | 8 | $ 1,012.00 |
| Lead Engineer | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | 1 | 8 | $ 988.40 |
| Architect 1 | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | 1 | 4 | $ 494.20 |
| Software Development Lead | GS-15 | $ 154.14 | 1 | 8 | $ 1,233.12 | 1 | 160 | $ 24,662.40 | 1 | 8 | $ 1,233.12 |
| Software Developer 1 | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | 1 | 4 | $ 494.20 |
| Software Developer 2 | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | 1 | 4 | $ 494.20 |
| Software Developer 3 | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | 1 | 4 | $ 494.20 |
| Software Developer 4 | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | 1 | 4 | $ 494.20 |
| Integration Lead | GS-13 | $ 123.55 | | | $ - | 1 | 80 | $ 9,884.00 | 1 | 4 | $ 494.20 |
| Integrator 1 | GS-11 | $ 108.08 | | | $ - | 1 | 80 | $ 8,646.40 | | | $ - |
| Test Lead | GS-13 | $ 126.50 | | | $ - | 1 | 160 | $ 20,240.00 | 1 | 4 | $ 506.00 |
| Tester 1 | GS-11 | $ 108.08 | | | $ - | 1 | 160 | $ 17,292.80 | 1 | 4 | $ 432.32 |
| Human Factors Engineer 1 | GS-13 | $ 123.55 | | | $ - | 1 | 80 | $ 9,884.00 | 1 | 4 | $ 494.20 |
| Cybersecurity 1 | GS-13 | $ 126.50 | | | $ - | 1 | 80 | $ 10,120.00 | 1 | 4 | $ 506.00 |
| Configuration Management (CM) | GS-11 | $ 108.08 | | | $ - | 1 | 160 | $ 17,292.80 | | | $ - |
| Trainers | GS-11 | $ 108.08 | | | $ - | 1 | 80 | $ 8,646.40 | | | $ - |
| Subject Matter Expert 1 | GS-13 | $ 123.55 | | | $ - | 1 | 160 | $ 19,768.00 | | | $ - |
| Subject Matter Expert 2 | GS-15 | $ 154.14 | | | $ - | 1 | 160 | $ 24,662.40 | | | $ - |
| | | | 2 | 16 | $ 2,466.24 | 19 | 2496 | $ 312,413.44 | 14 | 68 | $ 8,753.80 |
| | | | Avg | 8.0000 | | Avg | 137.7778 | | Avg | 4.8571 | |
| | | | Stdev | 0.0000 | | Stdev | 36.8711 | | Stdev | 1.7033 | |


Inputs
Number, Hourly Rate, Allocated Hours for Each Personnel Role
Action Performed
Monte Carlo: 1000 trials
Output
Mean Duration in Days & Stdev
Mean Cost & Stdev
16 Personnel Roles
Project Manager
Product Owner
Lead Systems Engineer
SW Architects
SW Development Lead
SW Developers
Integration Lead
Integrators
Test Lead
Testers
Human Factors Engineers
Cybersecurity Engineers
Configuration Mgmt Specialists
Training Specialists
Subject Matter Experts
System User Reps
Perform Sprint Planning, Execution, and Review D.1.3.3
Status
Perform Sprint Planning, Execution, & Review 1000 / 1000
Duration
Mean 15.96 Days
SD 2.62 Days
Cost
Mean $254434.39
SD $20131.33

| | | | Perform Sprint Planning, Execution and Review (D.1.3.3) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Update Product Backlog (D.1.3.3.1) | | | Prioritize Backlog (D.1.3.3.2) | | | Sprint Planning (D.1.3.3.3) | | | Sprint Review (D.1.3.3.7) | | | Sprint Retrospective (D.1.3.3.8) | | |
| Role | GS-Equiv | Hourly Rate | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost |
| Project Manager | GS-15 | $ 154.14 | | | $ - | 1 | 2 | $ 308 | | | $ - | 1 | 2 | $ 308 | | | $ - |
| Product Owner | GS-13 | $ 126.50 | | | $ - | 1 | 4 | $ 506 | | | $ - | 1 | 2 | $ 253 | | | $ - |
| Lead Engineer | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 4 | $ 494 | 1 | 8 | $ 988.40 | 1 | 8 | $ 988 | 1 | 3 | $ 371 |
| Architect 1 | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | | | $ - |
| Architect 2 | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - | 1 | 4 | $ 494 | | | $ - |
| Software Dev Lead | GS-15 | $ 154.14 | 1 | 16 | $ 2,466 | 1 | 4 | $ 617 | 1 | 16 | $ 2,466.24 | 1 | 8 | $ 1,233 | 1 | 3 | $ 462 |
| Software Developer 1 | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 2 | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 3 | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 4 | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 5 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 6 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 7 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 8 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 9 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 10 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 11 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 12 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 13 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 14 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 15 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 16 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 17 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 18 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 19 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |
| Software Developer 20 | GS-13 | $ 123.55 | | | $ - | | | $ - | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 1 | $ 124 |

| | | | Perform Sprint Planning, Execution and Review (D.1.3.3) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Update Product Backlog (D.1.3.3.1) | | | Prioritize Backlog (D.1.3.3.2) | | | Sprint Planning (D.1.3.3.3) | | | Sprint Review (D.1.3.3.7) | | | Sprint Retrospective (D.1.3.3.8) | | |
| Role | GS-Equiv | Hourly Rate | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost | # of FTEs | Hours Allocated per FTE | Projected Cost |
| Integration Lead | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 8 | $ 988.40 | 1 | 4 | $ 494 | 1 | 3 | $ 371 |
| Integrator 1 | GS-11 | $ 108.08 | 1 | 8 | $ 865 | | | $ - | 1 | 4 | $ 432.32 | 1 | 2 | $ 216 | 1 | 1 | $ 108 |
| Integrator 2 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 3 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 4 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 5 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 6 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 7 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 8 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 9 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Integrator 10 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | | | $ - | 1 | 1 | $ 108 |
| Test Lead | GS-13 | $ 126.50 | | | $ - | 1 | 2 | $ 253 | 1 | 4 | $ 506.00 | 1 | 2 | $ 253 | | | $ - |
| Tester 1 | GS-11 | $ 108.08 | | | $ - | 1 | 2 | $ 216 | 1 | 4 | $ 432.32 | 1 | 2 | $ 216 | | | $ - |
| Tester 2 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | 1 | 2 | $ 216 | | | $ - |
| Human Factors Eng 1 | GS-13 | $ 123.55 | 1 | 8 | $ 988 | 1 | 2 | $ 247 | 1 | 4 | $ 494.20 | 1 | 3 | $ 371 | | | $ - |
| Human Factors Eng 2 | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - | 1 | 3 | $ 371 | | | $ - |
| Cybersecurity 1 | GS-13 | $ 126.50 | 1 | 8 | $ 1,012 | 1 | 2 | $ 253 | 1 | 4 | $ 506.00 | 1 | 2 | $ 253 | | | $ - |
| Cybersecurity 2 | GS-13 | $ 126.50 | | | $ - | | | $ - | | | $ - | 1 | 2 | $ 253 | | | $ - |
| Configuration Mgmt (CM) | GS-11 | $ 108.08 | 1 | 4 | $ 432 | | | $ - | | | $ - | 1 | 2 | $ 216 | | | $ - |
| CM 2 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | 1 | 2 | $ 216 | | | $ - |
| Trainers | GS-11 | $ 108.08 | 1 | 4 | $ 432 | | | $ - | | | $ - | 1 | 2 | $ 216 | | | $ - |
| System User | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - | 1 | 2 | $ 216 | | | $ - |
| Subject Matter Expert 1 | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - | | | $ - | | | $ - |
| Subject Matter Expert 2 | GS-15 | $ 154.14 | | | $ - | | | $ - | | | $ - | | | $ - | | | $ - |


Inputs
Number, Hourly Rate, Allocated Hours for Each Personnel Role
Action Performed
Monte Carlo: 1000 trials
Output
Mean Duration in Days & Stdev
Mean Cost & Stdev
4 Personnel Roles
Configuration Mgmt Lead
CM Specialists
SW Development Lead
Lead Engineer
Receive SW Delivery and Documentation D.1.3.4
Status
Receive SW Delivery & Documentation
1000 / 1000
Duration
Mean 5.36 Days
SD 3.32 hours
Cost
Mean $8351.06
SD $659.40

| Receive Software Delivery and Documentation (D.1.3.4) | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Verify Delivery of License Agreements (D.1.3.4.1.1) | | | Verify Delivery Intellectual Property Rights (D.1.3.4.1.2) | | | Verify Delivery SW Build Instructions (D.1.3.4.1.3) | | | Verify Delivery Anti Virus Scans (D.1.3.4.1.4) | | |
| Position | GS-Equiv | Hourly Rate | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost |
| CM Lead | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - | | | $ - |
| CM Specialist 1 | GS-11 | $ 108.08 | 1 | 10 | $ 1,080.80 | 1 | 2 | $ 216.16 | 1 | 3 | $ 324.24 | 1 | 3 | $ 324.24 |
| CM Specialist 2 | GS-11 | $ 108.08 | 1 | 10 | $ 1,080.80 | 1 | 2 | $ 216.16 | 1 | 3 | $ 324.24 | 1 | 3 | $ 324.24 |

| | | | Verify Delivery Source Code (D.1.3.4.1.5) | | | Verify Delivery Required Scripts (D.1.3.4.1.6) | | | Verify Delivery SW Build Dependencies (D.1.3.4.1.7) | | | Verify Delivery Requirements Docs (D.1.3.4.1.8) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Position | GS-Equiv | Hourly Rate | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost |
| CM Lead | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - | 1 | 2 | $ 247.10 |
| CM Specialist 1 | GS-11 | $ 108.08 | 1 | 2 | $ 216.16 | 1 | 1 | $ 108.08 | 1 | 1 | $ 108.08 | | | $ - |
| CM Specialist 2 | GS-11 | $ 108.08 | 1 | 2 | $ 216.16 | 1 | 1 | $ 108.08 | 1 | 1 | $ 108.08 | | | $ - |

| | | | Verify Delivery SW Design Docs (D.1.3.4.1.9) | | | Verify Delivery Interface Requirements (D.1.3.4.1.10) | | | Verify Delivery Test Plans and Reports (D.1.3.4.1.11) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Position | GS-Equiv | Hourly Rate | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost |
| CM Lead | GS-13 | $ 123.55 | 1 | 1 | $ 123.55 | 1 | 2 | $ 247.10 | | | $ - |
| CM Specialist 1 | GS-11 | $ 108.08 | | | $ - | | | $ - | 1 | 6 | $ 648.48 |
| CM Specialist 2 | GS-11 | $ 108.08 | | | $ - | | | $ - | 1 | 6 | $ 648.48 |

| | | | Issues With Delivery (D.1.3.4.2) | | | Developer Receives Notifications (D.1.3.4.15/16, 18/19/20) | | | Lead Engineer Receives Notifications (D.1.3.4.26/27) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Position | GS-Equiv | Hourly Rate | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost |
| CM Lead | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - |
| CM Specialist 1 | GS-11 | $ 108.08 | 1 | 2 | $ 216.16 | | | $ - | | | $ - |
| CM Specialist 2 | GS-11 | $ 108.08 | | | $ - | | | $ - | | | $ - |
| SW Development Lead | GS-15 | $ 154.14 | | | $ - | 1 | 0.5 | $ 77.07 | 1 | 0.5 | $ 77.07 |
| Lead Engineer | GS-13 | $ 123.55 | | | $ - | | | $ - | | | $ - |

| | | | Lead Engineer Considers Waiver (D.1.3.4.22/24/25) | | | Accept or Reject Delivery & Notify Dev. (D.1.3.4.6/7/8, or 9/10, or 11/12/13) | | |
|---|---|---|---|---|---|---|---|---|
| Position | GS-Equiv | Hourly Rate | # of FTEs | Hours per FTE | Cost | # of FTEs | Hours per FTE | Cost |
| CM Lead | GS-13 | $ 123.55 | | | $ - | 1 | 1.5 | $ 185.33 |
| CM Specialist 1 | GS-11 | $ 108.08 | | | $ - | | | $ - |
| CM Specialist 2 | GS-11 | $ 108.08 | | | $ - | | | $ - |
| SW Development Lead | GS-15 | $ 154.14 | | | $ - | | | $ - |
| Lead Engineer | GS-13 | $ 123.55 | 1 | 4.25 | $ 525.09 | | | $ - |

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES


Agile Alliance. 2018. "12 Principles behind the Agile Manifesto." Accessed August 23, 2018. https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/.

Auguston, M. 2009. Software Architecture Built from Behavior Models. *ACM SIGSOFT Software Engineering Notes* 2009, *34*, 1–15.

Auguston, M. 2018. System and Software Architecture and Workflow Modeling Language Manual (Version 3.5). Accessed August 23, 2018. https://wiki.nps.edu/display/MP/Documentation.

Blanchard, Benjamin S., and Wolter J. Fabrycky. 2011. *Systems Engineering and Analysis.* 5th Ed. Upper Saddle River, NJ: Pearson Prentice Hall.

Boehm, Barry W. 1988. "A Spiral Model of Software Development and Enhancement." *Computer*, 21, no 5 (May 1988). doi:10.1109/2.59.

Boehm, Barry W., Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, and Bruce Steece. 2000. *Software Cost Estimation with COCOMO II.* Upper Saddle River, NJ: Pearson Prentice Hall.

Buede, Dennis M. 2009. *The Engineering Design of Systems: Models and Methods*. 2nd Ed. Hoboken, NJ: John Wiley & Sons.

Crawley, Edward, Bruce Cameron, and Daniel Selva. 2017. *System Architecture*. Hoboken, NJ: Pearson.

Cao, Lan, Ramesh, B., and Abdel-Hamid, T. 2010. "Modeling Dynamics in Agile Software Development." *ACM Transactions on Management, Information Systems*, 1, no. 1, article 5 (December): 5:1-5:26. doi:10.1145/1877725.1877730.

De Silva, Ian J., Heimdahl, Mats P. E., and Sanjai Rayadurgam. 2015. "A Reference Model for Simulating Agile Processes." *Department of Computer Science and Engineering University of Minnesota.* https://www.umsec.umn.edu/sites/www.umsec.umn.edu/files/desilva_rayadurgam_heimdahl-referenceModelForSimulatingAgileProcesses.pdf.

Douglass, Bruce P. 2016. Agile Systems Engineering. Elsevier. Accessed May 26, 2018. https://app.knovel.com/hotlink/toc/id:kpASE00001/Agile-systems-engineering/Agile-systems-engineering.

Federal Acquisition Regulations (FAR). 2016. Part 10. Market Research. Accessed May 27, 2018 https://www.acquisition.gov/sites/default/files/current/far/pdf/FAR.pdf.

Giammarco, Kristen 2012. "Architecture Model Based Interoperability Assessment." PhD diss., Naval Postgraduate School, Monterey, CA.

Giammarco, Kristin, and Kathleen Giles. 2018. "Verification and validation of behavior models using lightweight formal methods." *Disciplinary convergence in systems engineering research*. (August 2018): 431–447. doi: 10.1007/978-3-319-62217-0_31.

Glaiel, Firas, Allen Moulton, and Stuart Madnick. 2013. "A System Dynamics Investigation of Agile Software Development Methods." Working paper, Massachusetts Institute of Technology (MIT) Composite Information Systems Laboratory (CISL) Sloan School of Management, https://ic3.mit.edu/wp-content/uploads/2013-05.pdf.

Hayes, Will, and Wrubel Miller. 2017. Agile in Government: A Research Agenda for Agile Software Development. Accessed August 11, 2017. http://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_495976.pdf.

INCOSE. 2007. Systems Engineering Vision 2020, version 2.03. Seattle, WA, US: International Council on Systems Engineering (INCOSE), INCOSE-tp-2004-004-02.

———. 2011. *Systems Engineering Handbook: A Guide for System Life Cycle Process and Activities*, version 3.2.2. San Diego, CA, US: International Council on Systems Engineering (INCOSE), INCOSE-tp-2003-002-03.2.2.

Kellner, Marc I., Raymond J. Madachy and David M. Raffo. 1999. "Software Process Simulation Modeling: Why? What? How?" *Journal of System and Software*, 46, no 2/3 (April 1999). doi:10.1016/S0164-1212(99)00003-5.

Langford, Gary O. 2012. *Engineering Systems Integration: Theory, Metrics, and Methods*. Boca Routon, Florida: CRC Press.

Life cyclemodeling.org. 2015. Life cycle Modeling Language (LML) Specification 1.1. Accessed July 12, 2018. http://www.life cyclemodeling.org/spec/LML_Specification_1_1.pdf.

Madachy, Raymond J. 2008. *Software Process Dynamics*. Piscataway, NJ.: Wiley-IEEE.

Madachy, Raymond J., and Houston, Daniel. 2018. *What Every Engineer Should Know About Modeling and Simulation.* Boca Raton, FL: CRC Press.

Maier, Mark W. and Eberhardt Rechtin. 2009. *The Art of Systems Architecting*. 3rd Ed. Boca Raton, FL: CRC Press.

Moulton, Allen, Sean Ricks, John James, Greg Love, Aleksandra Markina-Khusid, Greg Howard, Paula Mahoney, Stuart Madnick. 2017. “Managerial Implications and Comparative Effects of SAFe Scaled Agile Methods in Government Software Acquisition.” Paper presented at 35th International Conference of the System Dynamics Society and 60th Anniversary of System Dynamics Celebration Cambridge, MA, USA, 16–20 (July), 2017. https://www.systemdynamics.org/assets/conferences/2017/proceed/index.html.

Mun, Johnathan. 2015. *Readings in Certified Quantitative Risk Management (CQRM): Applying Monte Carlo Risk Simulation, Strategic Real Options, Stochastic Forecasting, Portfolio Optimization, Data Analytics, Business Intelligence, and Decision Modeling*. Dublin, CA: Thompson-Shore, ROV Press, and IIPER Press.

Osmundson, John S. and Kristin M. Giammarco. 2017. SI4022 Lectures. Accessed May 26, 2017. https://wiki.nps.edu/pages/viewpage.action?title=SI4022+M07+Information+System+Architectures&spaceKey=MP.

Pressman, Roger S. 2010. *Software Engineering: A Practitioner’s Approach, 7th Ed*. New York, NY; McGraw-Hill.

Selby, Richard W. 2007. “Software Engineering: Barry W. Boehm’s Lifetime Contributions to Software Development, Management, and Research.” Hoboken, NJ; Wiley-Interscience.

Software Engineering Institute. 2017. *Learning about Agile in Government Settings.* Pittsburgh, PA: Carnegie Mellon University.

SPAWAR Systems Center Pacific (SSC Pacific). 2016. *Rapid Integration and Test Environment Standard Operating Procedures (SOP).* San Diego, CA: SPAWAR Systems Center Pacific.

SPEC Innovations. 2018. Innoslate 101. Accessed June 6, 2018. https://help.innoslate.com/how-to-guides/innoslate-101/.

———. 2017. Innoslate, 3.9. Manassas, VA. Accessed March 23, 2018. https://app.innoslate.com/project/p6X19PH/dashboard.

Standish Group. 2015. CHAOS Report. Accessed May 20, 2017. https://www.cs.nmt.edu/~cs328/reading/Standish.pdf.

Systems and Software Consortium. 2007. *Managing Agile Software Development SPC-2002007 Version 4.5.* Herndon, VA: Systems and Software Consortium.

Wilson, Rosa, Raymond Madachy, Bradford Clark, and Barry Boehm. 2017. “Early Phase Cost Models for Agile Software Processes in the U.S. DoD.” Paper presented at *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement,* Toronto, ON, Canada, November, 2017. doi:10.1109/ESEM.2017.10.

White, A. S. 2014. “An Agile Project System Dynamics Simulation Model.” *International Journal of Information Technologies and Systems Approach (IJITSA)* 7, no. 1 (2014): 55–79. doi:10.4018/ijitsa.2014010104.

Wrubel, Eileen, and Jon Gross. 2015. “Contracting for Agile Software Development in the Department of Defense: An Introduction.” Accessed August 11, 2017. http://resources.sei.cmu.edu/asset_files/TechnicalNote/2015_004_001_442515.pdf.

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California