SYM-AM-24-106

Excerpt from the Proceedings of the Twenty-First Annual Acquisition Research Symposium

Acquisition Research: Creating Synergy for Informed Change

May 8–9, 2024

Published: April 30, 2024

Capability-Based Software Cost Estimation (CaBSCE): Proposing a New Method to Estimate Software Costs

Anandi Hira—leads research to advance the state of the art in software cost estimation and supports programs with the resulting estimation and measurement processes in her current role at Carnegie Mellon University's (CMU's) Software Engineering Institute (SEI). Her most recent research objective is to develop a new, innovative, capability-based software cost estimation model. Hira received her PhD in software cost estimation under Barry Boehm at the University of Southern California (USC), where she collected software development data and calibrated the COCOMO® II software cost estimation model for Function Points. [info@sei.cmu.edu]

Abstract

The dynamic nature of software has led to exciting technology improvements as well as challenges in cost estimation. Defining specific software requirements early in the lifecycle is impossible. The Software Acquisition Pathway (SWP), defined in Department of Defense (DoD) Instruction 5000.87 (Office of the Under Secretary of Defense for Acquisition and Sustainment, 2020), requires only a high-level specification of needed capabilities—a capability needs statement (CNS) or software initial capabilities document (SW-ICD)—before entering the execution phase of software-intensive programs (Defense Acquisition University, 2020b). Most cost estimators for the U.S. government rely on size-based estimation methods, primarily source lines of code (SLOC). However, estimating SLOC requires understanding specific implementation details and can only be accurately estimated near program completion. The software cost estimation community needs a capability-based estimation method that aligns with the SWP and meets the government's need for evidence-based, flexible, and defensible estimates (Defense Acquisition University, 2020a). In response to these competing needs, the Software Engineering Institute (SEI) proposes developing a software capability-based estimation model requiring only high-level information. Using existing software development effort data, the SEI plans to analyze descriptive fields to cluster data points based on similar software functions and complexity. The resulting model would provide evidence-based rough order of magnitude (ROM) estimates for the initial stages of identifying required capabilities.

Introduction

Developers and managers from the commercial space, private sector, and government acquisition learned that defining software requirements in advance is an impossible expectation (Highsmith, 2002; Raza & Waheed, 2018). Requirements change significantly because both users and customers might not have a clear understanding of what they want and need, different stakeholders interpret requirements differently, and quickly evolving technology affects the needs of the new project (De Lucia & Qusef, 2010). These requirements issues have caused many software projects to either fail or significantly overrun their budgets and/or schedules (Shah & Patel, 2014). Several software engineers came together to draft the Agile Manifesto and its corresponding 12 principles to improve software development. Agile software development primarily addresses requirements issues by "[w]elcom[ing] changing requirements, even late in development," showing progress, and giving customers and users the opportunity to re-evaluate requirements. This is done through short increments of working software (Beck et al., 2001). Allowing changes in requirements improves the probability of success but poses challenges for estimators. Without knowing exactly what the final product will be able to do, estimators cannot accurately estimate the program's effort, cost, and schedule required to develop what the customers and users need. In the commercial industry, many organizations have circumnavigated this issue by implementing high-priority requirements and addressing as many requirements as the customers can afford. Since developers demonstrate working code at each increment, they continue working until the customer is sufficiently satisfied (Milani & Rossi, 2020). However, estimates play an essential role in the U.S. government's defense acquisition process, as estimates determine which programs to authorize and how to distribute annual funds across active programs.

Traditionally, the government identifies high-level capability needs, which are broken down into detailed requirements. These requirements serve as a basis for developing cost estimates that determine the feasibility of the solution and allow the comparison of alternate options. If the government both authorizes and appropriates funds for the program, the acquiring organization typically posts a request for proposal package (RFP), and organizations may place their bids towards the contract. These organizations must provide their own estimated cost and schedule, which the government evaluates to award the best-value contractors. After the contract is awarded and implementation starts, the program's estimates are updated periodically to ensure that the updated estimates reflect the learning and changes in the scope. The government encourages contractors to adopt Agile and DevSecOps (DevOps with an added focus on security, which is commonly used by defense programs) practices to "rapidly adapt to emerging user needs" and "shorten acquisition timelines" (Coonce & Alleman, 2017) and "match the speed at which new IT capabilities are being introduced in today's information age" (Defense Science Board, 2009). However, traditional waterfall acquisition lifecycle does not align with Agile and DevSecOps practices. In response, the government implemented the Software Acquisition Pathway (SWP) defined in DoD Instruction 5000.87 (Office of the Under Secretary of Defense for Acquisition and Sustainment, 2020). The SWP allows moving into the execution phase with CNS or SW-ICD, which provide high-level operational capabilities, enhancements to existing capabilities, features, interoperability needs, legacy interfaces, and other attributes relevant to define how the software solution relates to the overall threat environment (Office of the Under Secretary of Defense for Acquisition and Sustainment, 2020). Detailed requirements evolve through Agile increments. However, cost estimates are still required during the planning phase for the government's budget process. Hence, defense acquisition programs struggle with developing estimates and providing evidence to defend their budget requests.

The SEI proposes developing a capability-based software cost estimation model (CaBSCE) based on high-level information that corresponds to the capabilities provided at program initiation. CaBSCE would accommodate uncertainty in later architecture, reuse, and implementation decisions. The underlying sizing method is based on identifying clusters of software functions in common capabilities, and the cost model is calibrated with historical data from comparable efforts, thus satisfying the government's need for flexible, credible, and defensible estimates (Defense Acquisition University, 2020a) as well as the estimators' need for a cost model based on capabilities. The next section defines capability-based planning (CBP) and capabilities, which explains the level of detail made available when early estimates are needed, followed by quick summaries of predominant and relevant software cost estimation techniques and a description of how the SEI will develop CaBSCE.

Background

Capability-Based Planning (CBP)

Defining CBP

Several countries' governments (e.g., United States, Canada, Australia) have started to use CBP to "design an appropriate [military] force" and one that is "postured to adequately deal with the challenges of the future" (Taliaferro et al., 2019). Instead of focusing on who an adversary might be, where a war might occur, or how an adversary might fight (Biltgen, 2007; Hanley et al., 2006; Planeaux, 2003; Walker, 2005), the "goal is to plan for robust, flexible forces, capable of meeting a wide variety of threats, rather than an 'optimal' force for a narrow set of threats" (Titus, 2004). The "objective is to develop a flexible, adaptable, robust, and sustainable (i.e., technically manageable and financially affordable) force structure postured to address all the challenges associated with a nation's strategic defense and security environment, considering budgets and uncertainty" (Taliaferro et al., 2019). With CBP, governments and defense departments evaluate "the development and evolution of capabilities, rather than specific programs or function" (Webb, 2008) to go "from programs to portfolios of capabilities" (Bexfield & Disbrow, 2004) and "determine an efficient and effective mix of military forces" (Taliaferro et al., 2019). A capability serves as a goal that enables us to decide whether a specific technology or process supports achieving that goal (Taliaferro et al., 2019). In summary, CBP underscores the significance of cultivating adaptable military capabilities and addressing future challenges in national defense by transcending the limitations of specific defense organizations, military services, programs, or functions.

The most referenced definition of CBP comes from Paul Davis (2002):

> *Capabilities-based planning is planning, under uncertainty, to provide capabilities suitable for a wide range of modern-day challenges and circumstances, while working within an economic framework.*

Essentially, CBP must "confront—rather than discount—uncertainty, to express risk in meaningful terms, and to weigh costs and benefits simultaneously" (Committee on Naval Analytical Capabilities and Improving Capabilities-Based Planning, 2005). Uncertainty stems from two sources: (1) the scenarios that describe the needed capabilities and (2) the "details of assumptions in those scenarios" (Committee on Naval Analytical Capabilities and Improving Capabilities-Based Planning, 2005). Additionally, CBP should consider a diverse "range of competitive options and trade-offs before making the choices necessitated by a budget" (Committee on Naval Analytical Capabilities and Improving Capabilities-Based Planning, 2005). In other words, CBP focuses on goals and outcomes and encourages innovation (Anastasios, 2014; Bexfield & Disbrow, 2004; Chim et al., 2010; Desgagné, 2009; Technical Cooperation Program, 2004) by "moving away from determining equipment solutions prematurely" (Technical Cooperation Program, 2004). This "provides a means to compare different options for achieving the same capability" (Technical Cooperation Program, 2004). Hence, it is not CBP's goal to "engineer planning processes to a fine level of detail, but rather to design an effective decision-support mechanism for regular, rigorous integration of planning process outputs" (Hanley et al., 2006). To support this high-level, generalizable framework, CBP "starts with a top-down definition through scenarios, case studies, or use cases" without eliciting detailed requirements (Alleman, 2020b).

The CBP processes require stakeholders to "think broadly about the entire scenario space of possibilities" (Davis et al., 2008). "Scenarios provide the essential link between defense policy and capability objectives" (Technical Cooperation Program, 2004). Broad, high-level scenarios help with the following:

- develop "realistic capability goals" (Technical Cooperation Program, 2004)
- provide "context and a means to share assumptions" (Hales & Chouinard, 2011) and therefore "facilitate communication" as it becomes "easier to compare options in a strategic-level framework if everyone has a fairly concrete mental image of what the evaluation cases are" (Davis et al., 2008)
- provide context for capability assessment (Titus, 2004) and identifying gaps associated with the mission area (Chairman of the Staff, 2009)
- "provide a way to test the concept against the breadth of the defense strategy" and "the spectrum of conditions to be considered" (Chairman of the Staff, 2009)
- "assess whether the capability at issue would merely be nice to have logically or would make a difference in plausible cases of concern" (Davis et al., 2008)

Therefore, to make strategic and successful decisions, governments must evaluate their ideal and existing capabilities at the highest level possible while refraining from making decisions that can overlook uncertainty, risk, and budget constraints. The generalizability of the CBP framework corresponds to the generalizable definitions of capabilities. CBP and planning at the capabilities level aim to offer a holistic and flexible perspective while ensuring success.

Defining Capabilities

Capabilities have been defined in many ways to support the high-level, flexible, and comprehensive CBP framework. The definitions can be grouped into three categories: (1) the description of objectives or high-level needs, (2) operational outcomes, and (3) the ability to produce or achieve some type of outcome.

Definitions in the first set collectively illustrate that capabilities represent objectives or high-level needs that form the fundamental basis of strategic decision making:

- "Functional approach to the articulation of broad requirements without necessarily specifying the resources that may be involved" (Chim et al., 2010)
- "Defined by an operational user and expressed in broad operational terms" (Iacobucci, 2012)
- "Foundation of defining the technical and operational requirements of the product or service produced by the project. Without the defined capabilities, those requirements have no reason for being" (Alleman, 2020b).
- "Consist of far more than just technology; in fact, technology underpins just one element—materiel—of a capability" (Hanley et al., 2006)
- "Capabilities are not the same as features and functions; they enable demands to be met without the explicit specification of the solution" (Alleman, 2020a).
- "Capabilities provide the answer to the following question: To achieve our objectives, what capabilities must we possess?" (Davis et al., 2008). In other words, "to achieve our objectives," what must we be able to do? Capabilities include talent, expertise, materiel, and capacity, among other things.

The next category of capability definitions is related to the above, but instead of focusing on objectives and needs, it focuses on operational outcomes:

- “A description of the military operational output or outcome that a unit, force, or organization is able (and usually constituted or organized) to deliver” (Steele, 2021)
- “Define the future effects needed for agencies to meet their mission and transform into a more agile and adaptable force” (Kossakowski, 2005).
- “the combination of military equipment, personnel, logistics support, training, resources, etc. that provides Defence with the ability to achieve its operational aims” (Neaga et al., 2009)

While the first two categories of definitions focus on what can be changed or done, the last group of definitions focuses on the ability to produce or achieve some type of outcome rather than the outcome per se:

- “Ability to achieve a desired effect under specified standards and conditions through combinations of means and ways to perform a set of tasks” (Bexfield & Disbrow, 2004)
- “Further defined as the ability to contribute to the achievement of a desired effect in a given environment within a specified time and the sustainment of that effect for a designated period” (Steele, 2021)
- “Wherewithal to complete a task or produce an effect within a set of specified performance standards and environmental conditions” (Taliaferro et al., 2019)
- “The ability to achieve an objective in a military operation” (Chairman of the Staff, 2009)

At first, these definitions may seem divergent. However, they collectively describe the essence of capabilities as high-level descriptions essential for fulfilling the government’s objectives. These multifaceted definitions capture different dimensions in which achieving a solution must encompass. The individual definitions may represent a view from a time in the lifecycle, the viewer’s role, or a stakeholder’s value proposition.

Software Acquisition Pathway (SWP)

The output of CBP (i.e., identified capability needs) serves as the input to the defense acquisition lifecycle. In the waterfall acquisition lifecycle, the high-level capability needs are broken down into detailed requirements that specify how they need to be met. Traditionally, software was treated as a component of the overarching system or “an enabler of hardware systems and weapon platforms” (Defense Innovation Board, 2019). As the government realized that software is predominant and “defines our mission-critical capabilities and our ability to sense, share, integrate, coordinate, and act” (Defense Innovation Board, 2019), the government developed the SWP to provide specific guidance for software development and remove the roadblocks to Agile and DevSecOps software development practices in the defense acquisition programs. The SWP allows software development efforts to adapt quickly and meet capability needs; it also enables the “timely acquisition of custom software capabilities developed” (Office of the Under Secretary of Defense for Aquisition and Sustainment, 2020).

The SWP abandons a waterfall lifecycle model in favor of an incremental and iterative one—that is compatible with modern software development practices such as Agile and DevSecOps. The high-level capability needs replace detailed requirements before moving into the execution phase. In compliance with Agile practices, the requirements evolve through continuous user involvement and high priority needs are addressed earlier. In the execution phase, requirements engineering, development, integration, testing, and deploying software occur incrementally; programs are required to deliver software capability releases every year or more frequently. Cost estimates are still required before moving to the execution phase and are updated annually (Office of the Under Secretary of Defense for Aquisition and Sustainment, 2020). Initial estimates are not expected to precisely capture the required cost and schedule, but they should be updated iteratively as requirements, architecture, and implementation decisions are refined (Defense Acquisition University, 2020a).

Software Cost Estimation Methodologies

Cost Estimation Criteria

Estimation plays a crucial role in the decision-making processes of the CBP framework, since "the most effective and efficient options to satisfy the requirements" and fill capability gaps "are sought" (Davis, 2002). Estimation is particularly vital when both comparing and analyzing all possible and alternative solutions within CBP. Without estimates for the potential outcomes of each choice in the list of alternatives, it becomes impossible to determine which capabilities are best suited to meet the mission's needs and which ones fulfill a business case. As Alleman (2020) emphasizes, credible decisions in the face of uncertainty rely on estimates. Therefore, the accuracy and reliability of estimates are critical to ensuring that CBP can effectively inform decisions about which capabilities to pursue, aligning them with strategic objectives and mission requirements. The U.S. government also requires cost estimates to justify the costs of a program and distribute annual funds across active programs. For estimates to effectively support CBP and the U.S. government's budget processes, the estimation method/model must meet the following criteria (the labels in the beginning correspond to labels used in Table 1):

1. Early Lifecycle: be applicable to high-level capabilities early in the acquisition lifecycle—before contractors and developers see the capability needs or requirements
2. Defensible: be defensible and evidence-based (based on historical data)
3. Generalizable: be generalizable, as the underlying cost models should be contractor agnostic (i.e., estimates should be applicable before contractors have been selected)
4. Cost Analysis: be able to perform cost analysis and answer questions such as, How much will it cost, and how long will it take to reach a minimal viable product (MVP)? What would a minimum viable capability release (MVCR) entail? Will required capabilities be completed by a deadline? How will the schedule and/or cost be affected by a reduced budget?

An estimate's inability to meet the above criteria can lead to the government cutting a program's budget, which will delay capability delivery or require descoping of capabilities. Unfortunately, as demonstrated in this section, none of the predominant, existing software cost estimation methods meet all the above criteria.

Source Lines of Code (SLOC)

Most cost estimators for government agencies still rely on size-based estimation, primarily using SLOC, due to its quantifiability, high correlation with effort (Albrecht & Gaffney, 1983; Kemerer, 1987), and large repository of historical data and cost estimation models. But estimating SLOC requires understanding specific implementation details, leading to significant changes in SLOC estimates throughout a program's lifecycle (GAO, 2020). SLOC can only be accurately estimated when a program is near completion (Boehm, 1981). In summary, SLOC fails to meet criterion 1—being applicable to high-level capabilities early in the lifecycle.

Story Points

Story Points are often used in Agile environments by the development team to (based on their judgment or subjective comparison to previous tasks) estimate the level of difficulty and required effort to implement the requirements. Development teams use numbers, for example, from the Fibonacci sequence, to signify the difficulty and required effort to implement requirements expressed as user stories (Cohn, 2004). However, Story Points were not developed to estimate effort or costs; they are part of an exercise used by the development team to determine which requirements they can tackle within the capacity of a development sprint. In several experiments, Jørgensen (2004) found that Story Points estimates are subject to bias with respect to the order in which projects are estimated. Jørgensen found that estimators tend to estimate larger tasks better after estimating smaller tasks. After estimating larger tasks, estimators are more likely to give optimistic (less effort) estimates for smaller tasks. Additionally, estimators may have the tendency to think that of two similarly sized projects, the second one (despite the order in which the projects are evaluated) will seem larger than the first one (Jørgensen, 2004). Since Story Points are estimated by the development team, they would not be available until the developers received the requirements of the system, which is later in the acquisition lifecycle. Hence, Story Points fail to meet criteria 1, 2, and 3.

Analogy-Based Estimation

In analogy-based estimation, program managers and subject matter experts (SMEs) typically use high-level characteristics, such as the type of program (e.g., information management system) or development process, to select an analogous program or data and apply a complexity factor to account for differences between the current and past programs (Ozkaya et al., 2008). While analogy-based estimation can be used early in the lifecycle, it still faces the same expert-based biases that Jørgensen (2004) described in his study. Therefore, analogy-based estimation fails to meet criterion 2 (i.e., being defensible and evidence-based).

Case-Based Reasoning

Academic research has studied case-based reasoning, but the studies focused on the variables available in the dataset (Idri et al., 2015; Shepperd & Schofield, 1997) rather than the functions of the program itself. Many studies found size “to be an influential factor” (Idri et al., 2015), while other studies focused on variables that would only be available or predictable at the time estimates are needed (Shepperd & Schofield, 1997). Some example variables used include programming language, number of input or output message types, and number of files changed as part of an enhancement task (Shepperd & Schofield, 1997). Given that an estimator would not know the size of the program being estimated and many of the example variables could not be used to identify similar programs across organizations, case-based estimation could not be applied early in the lifecycle (criterion 1) or be generalizable (criterion 3).

Number of Applications

The Air Force Cost Analysis Agency (AFCAA) led a study to develop cost models for DevSecOps programs that yielded cost estimating equations that can estimate a program’s fiscal year costs based on the number of applications that would be in development in that year (Bradshaw, 2022). However, these cost models do not estimate the total required effort and cost of the applications to produce the needed capabilities. Therefore, this method is unable to perform cost analysis and answer questions like how long it will take to develop a capability (criterion 4).

Function Points

Function Points represent the size of software based on its functional processes, which consist of inputs from users, reading from or writing to memory, and outputting results. Project stakeholders can define such functional processes early in the lifecycle and would provide objective sizing that is applicable across contractors and programs. With data, cost estimation models support cost analyses like SLOC-based cost estimation models. Function Points seem to be the most promising solution because they meet all the above criteria for cost estimates. However, building historical datasets and generalizable cost models applicable to the defense acquisition solution space would require years of data collection because sufficient data is not available at present.[1] Though the International Software Benchmarking Standards Group (ISBSG) provides software development data with Function Points, the dataset primarily represents the commercial industry. Additionally, since Function Points is based on functional processes, they do not account for nonfunctional requirements (e.g., security and reliability) or algorithmic complexity (Hira, 2020)—both of which can have significant effects on software development efforts.

Estimation Methods Summary

Table 1 summarizes the existing software cost estimation methods reviewed in this section and how they fail to meet the estimation criteria required to satisfy and be compatible with the CBP framework, the government's budgeting process, and the SWP acquisition lifecycle. While Function Points meet all the criteria, sufficient data does not exist within the defense acquisition solution space to develop generalizable cost estimation models. The current state of the art prevents defense programs from fully implementing and benefitting from modern Agile and DevSecOps software development practices.

Table 1. Summary of Predominant Software Cost Estimation Methods Failing to Meet Estimation Criteria for CBP and SWP

Criteria	SLOC	Story Points	Analogy-Based Estimation	Case-Based Reasoning	AFCAA DevSecOps	Function Points
Early Lifecycle	**No**	**No**		**No**		
Defensible		**No**	**No**			
Generalizable		**No**		**No**		
Cost Analysis					**No**	
Data Availability						**No**

[1] Publication Pending: Hira, A., & Kwok, B. (2024). (in press). What is the U.S. DoD cost estimation community saying about Agile? *Journal of Cost Analysis and Parametrics, 12*. When published, the journal will be accessible here: https://www.iceaaonline.com/publications/#journal

Proposed Solution: Capability-Based Software Cost Estimation (CaBSCE)

The CBP framework, the government's budgeting process, and the SWP acquisition lifecycle pose a difficult set of criteria for cost estimation methods to fulfill. Of the existing software cost estimation methods, analogy-based estimation, case-based reasoning, and Function Points are closest to fulfilling all the cost-estimation criteria. The SEI proposes taking aspects of each of these methods and developing a software cost estimation method that can meet all the estimation criteria—a capability-based software cost estimation method (CaBSCE). To develop CaBSCE, the SEI will identify clusters or groups (analogy-based estimation) of similar software functions (Function Points) calibrated with historical data from comparable efforts (case-based reasoning). For example, GPS/navigation components, despite the device they are installed on, must generally perform three functions: (1) identify the current location, (2) identify the destination, and (3) determine how to get from the current location to the destination with the use of trigonometry and artificial intelligence (AI) algorithms. Such functions would differ from compilers or text parsers, which peruse text either by comparing it to pre-specified patterns or identifying patterns in the text. This method would describe two different capability clusters, or groups, of similar software functions. These capability clusters would be a level lower than capabilities identified in CNSs and SW-ICDs (in terms of detail), but stakeholders involved at program initiation could identify the software functions required and consider alternate solutions at this level of software functions. (See Figure 1 for an example of a capability need and how software functions can be identified to satisfy it.) The associated effort ranges for each capability cluster serve as evidence-based ROM estimates. Figure 2 summarizes the proposed research methodology, and Table 2 describes how CaBSCE would satisfy the cost estimation method criteria identified in the previous section.

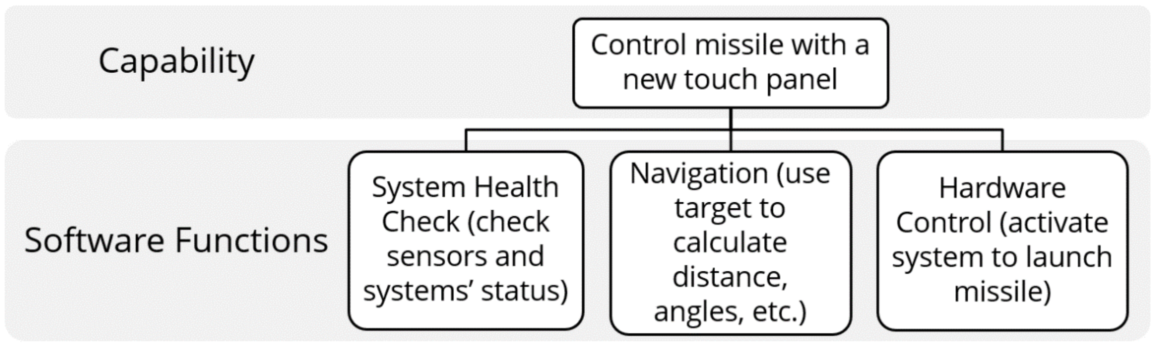

Capability
Control missile with a new touch panel
Software Functions
System Health Check (check sensors and systems' status)
Navigation (use target to calculate distance, angles, etc.)
Hardware Control (activate system to launch missile)

Figure 1. Example of a Capability Need and the Software Functions Required to Meet It

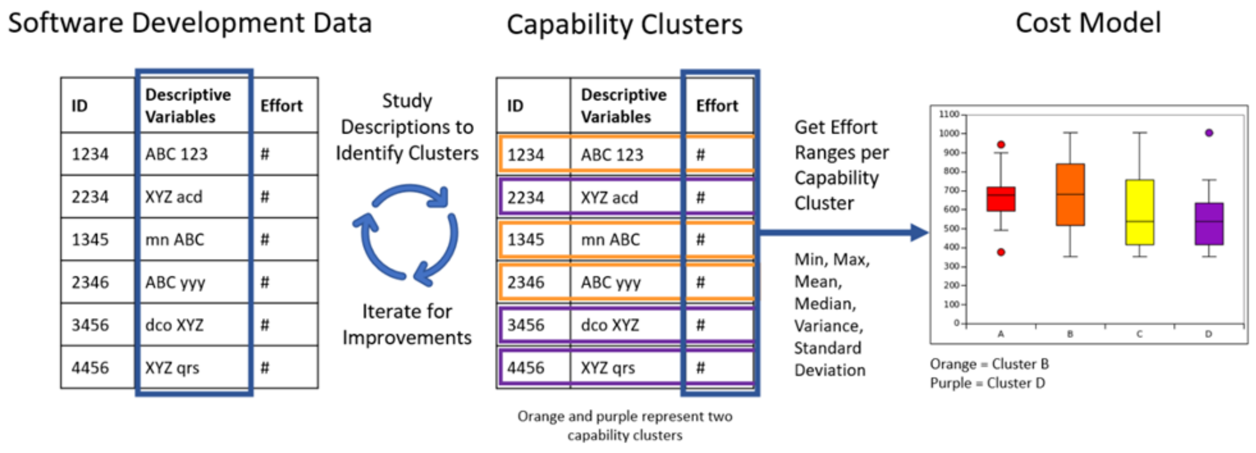

Software Development Data
Capability Clusters
Cost Model
ID
Descriptive Variables
Effort
1234
ABC 123
2234
XYZ acd
1345
mn ABC
2346
ABC yyy
3456
dco XYZ
4456
XYZ qrs
#
Study Descriptions to Identify Clusters
Iterate for Improvements
Get Effort Ranges per Capability Cluster
Min, Max, Mean, Median, Variance, Standard Deviation
Orange and purple represent two capability clusters
Orange = Cluster B
Purple = Cluster D

Figure 2. High-Level Research Methodology Used to Develop CaBSCE

Table 2. Description of How CaBSCE Meets the Cost Estimation Criteria

Criteria	How CaBSCE Meets the Criterion
Early Lifecycle	Though the exact solution or architectural and developmental requirements are not set yet, CaBSCE will be based on high-level software functions that can be identified within the CBP framework.
Evidence-Based	Using the clusters/groups of software functions, the effort ranges for each cluster/group will come from similar functions. The data points may represent different solutions, therefore providing an evidence-based yet capability-based estimate.
Generalizable	The datasets consist of software developed across many application types and organizations. Therefore, the resulting cost model would be generalizable across application types and organizations.
Cost Analysis	The goal is to get the required effort for completed software functions, which would allow cost estimators and program leadership to perform cost analysis.
Data Availability	The study will use existing software development datasets (ISBSG and other datasets), which provide total effort and descriptive variables for analysis.

The SEI will utilize existing software development datasets: the ISBSG and other datasets. The ISBSG dataset consists of thousands of projects that span several industries and business types and more than 32 countries. The dataset provides a few variables describing the organization type, industry, and application type. The SEI will analyze the descriptive information in the datasets to identify data points with similar software functions. (More details are provided in the next subsection.)

Classifying Software Functions

The Software Cost Estimation Methodologies section mentions several attempts to categorize data points to improve the ability to estimate in the software development lifecycle and improve estimation accuracy. Application types and domains are used to categorize software by the architecture/design typically followed (for example, client-server or database management). Many studies used application types and domains for clustering analysis (Van Hai et al., 2022) or as a categorical variable in cost estimation equations (Rosa et al., 2021). Coonce and Alleman (2017) hypothesize a standardized way of grouping the historical data on features and attributes that would support CBP and early lifecycle estimation for software. A drawback of using the application domains and types as similar and analogous data or projects is the inability to take advantage of new technology applied in another application domain or type. As an example, to estimate the first application of AI or machine learning (ML) in the satellite domain using analogy, estimators would use the most similar satellite program and add a complexity factor to account for the AI and ML application. While developing software for a satellite is evidence-based, the complexity factor or attempt to account for AI and ML would be subjective. Typically, estimators do not look outside their domain for historical data.

The SEI proposes identifying fundamental software functions that can span application domains and types. This way, expanding on the example of including AI and ML in a satellite program, estimators can pull from experience in other domains/types. This methodology can be simplified as doing a bottom-up, analogy-based estimation: Estimators and stakeholders would break down the fundamental software functions of capabilities and use analogous data for all identified functions to build up the total cost estimate. Boehm et al. (2000) developed a comprehensive definition for software product complexity that consists of five types of operations: control, computation, device-dependent, data management, and user interface management (replicated in Table 3), which the SEI will use to identify clusters or groups of application types by software functions using the following steps:

1. Extract unique application types from the datasets.
2. Classify the unique application types, using descriptive variables as guidance, across the five software operation types and levels of complexity.
3. Identify application types with similar combinations of complexities through clustering analysis.
4. Name the clusters/groups to describe the major functions and identify common words associated with each function, using the text in the descriptive variables. Essentially, the SEI will develop a dictionary to define software functions.
5. Apply the dictionary on the datasets and extract the effort ranges for each software functions cluster, which provides comparable efforts for similar functions, serving as the evidence-based ROM estimates.

Table 3. Software Product Complexity Description from Software Cost Estimation Model COCOMO® II (Boehm et al., 2000)

	Control Operations	**Computation Operations**	**Device-Dependent Operations**	**Data Management Operations**	**User Interface Management Operations**
Very Low	Straight-line code with few non-nested structured programming operations	Evaluation of simple expressions; example: A = B + C*(D-E)	Simple read, write statements with simple formats	Simple arrays in main memory	Simple input forms; report generators
Low	Straightforward nesting of structured programming operators; mostly simple predicates	Evaluation of moderate-level expressions; example: D = SQRT(B*2-4*A*C)	No cognizance needed of particular processor or I/O device characteristics	Single file subsetting with no data structure changes, no edits, and no intermediate files	Use of simple GUI builders

	Control Operations	Computation Operations	Device-Dependent Operations	Data Management Operations	User Interface Management Operations
Nominal	Mostly simple nesting; some inter-module controls; decision tables, simple callbacks, or message passing	Use of standard math and statistical routines; matrix/vector operations	I/O processing that includes device selection, status checking, and error processing	Multi-file input and single-file output; simple structural changes, simple edits	Simple use of widgets
High	Highly nested structured programming operators with many compound predicates; queue and stack control	Basic numerical analysis	Operations at the physical I/O level; optimized I/O overlap	Simple triggers activated by data stream contents	Widget development and extension; voice I/O; multimedia
Very High	Reentrant and recursive coding; fixed-priority interrupt handling; task sync, complex callbacks	Difficult but structured numerical analysis	Routines for interrupt diagnosis, servicing, and masking; communication line handling	Distributed database coordination; complex triggers; search optimization	Moderately complex 2D/3D, dynamic graphics, multimedia
Extra High	Multiple resource scheduling	Difficult and unstructured numerical analysis	Device timing-dependent coding; micro-programmed operations	Highly coupled, dynamic relational and object structures	Complex multimedia, virtual reality, natural language

Figure 3 demonstrates examples of step 2 (classify the unique application types, using descriptive variables as guidance, across the five software operation types and levels of complexity) for GPS/navigation and compiler/text parser. As mentioned earlier, GPS/navigation components, despite the device they are installed on, must generally perform three functions: (1) identify the current location, (2) identify the destination, and (3) determine how to get from the current location to the destination with the use of trigonometry and AI algorithms. This understanding translates to the COCOMO® II product complexity description like this:

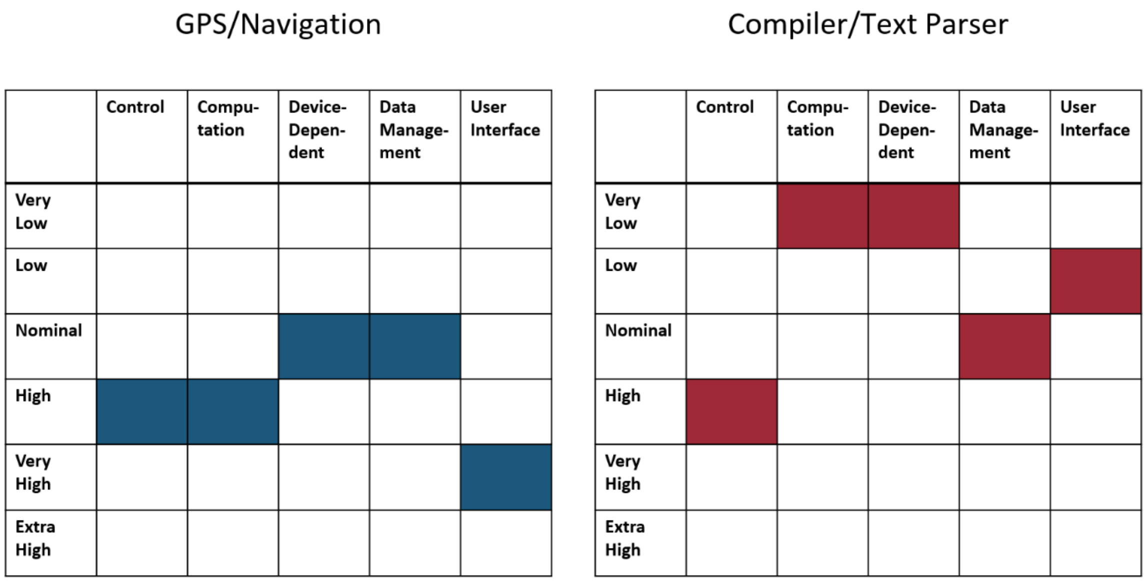
- Control Operations – High: AI algorithms to determine the best path to reach destination from current location will require nested programming, compound predicates, and queue and stack control.
- Computation Operations – High: calculating current and destination locations and paths between them requires numerical approximations and trigonometry.
- Device-Dependent Operations – Nominal: data to identify current location requires reading data from hardware (e.g., sensors).

- Data Management Operations – Nominal: data expressing current location, destination, and calculations for possible paths will require data structures.
- User Interface Management Operations – Very High: 2D/3D map displays for human-friendly operation.

On the other hand, compilers or text parsers peruse text either by comparing it to pre-specified patterns or identifying patterns in the text. This definition translates to the COCOMO® II product complexity descriptions as follows:

- Control Operations – High: comparing text to rules or identifying patterns in text requires nested programming, compound predicates, and queue and stack control.
- Computation Operations – Very Low: addition and subtraction may be needed to track patterns.
- Device-Dependent Operations – Very Low: simple reading and writing of inputs and outputs.
- Data Management Operations – Nominal: input and output data require data structures.
- User Interface Management Operations – Low: simple user interface is sufficient.

Step 3 then identifies application types with similar combinations of complexities through clustering analysis. For example, the software underlying scientific calculators may have similar complexity combinations as GPS/navigation. With this methodology, the SEI will define software functions with similar complexity that spans application domains.



GPS/Navigation

	Control	Compu-tation	Device-Depen-dent	Data Manage-ment	User Interface
Very Low					
Low					
Nominal			🟦	🟦	
High	🟦	🟦			
Very High					🟦
Extra High					

Compiler/Text Parser

	Control	Compu-tation	Device-Depen-dent	Data Manage-ment	User Interface
Very Low		🟥	🟥		
Low					🟥
Nominal				🟥	
High	🟥				
Very High					
Extra High					

Figure 3. GPS/Navigation (Left) and Compiler/Text Parser (Right) Application Types Mapped to COCOMO® II's Software Product Complexity Description

Conclusion

Software's dynamic nature has led to significant technology improvements in an ever-evolving environment, which contributes to changing requirements throughout the software development lifecycle. Agile software development welcomes changing requirements to improve customer satisfaction and project success. While traditional government defense acquisition processes required early comprehension of the requirements for estimates and budgets, the release of the SWP brought significantly more flexibility. The SWP shortens the software acquisition lifecycle and is compatible with both Agile's and DevSecOps' incremental learning and progress. High-level capabilities replace detailed requirements before starting development. In CBP, it is important to avoid defining specific implementation solutions for capabilities so that there is space for innovation and the most cost-effective solution can be determined. Unfortunately, this change presents unique and challenging constraints on software cost estimation needs. Existing software cost estimation methods depend on detailed requirements or do not fully utilize existing historical data.

The SEI proposes implementing CaBSCE, a software cost estimation method based on high-level descriptions of software functions and using comparable historical data for evidence-based ROM estimates. CaBSCE eliminates the weaknesses of existing software cost estimation methods while meeting the government's need for evidence-based, flexible, and defensible estimates. With CaBSCE, estimators will be able to identify analogous functions that map to the CNS or SW-ICD and use comparable efforts from historical data for defensible estimates that account for uncertainties. Programs will likely get less pushback from the government for budget requests, enabling efficient allocation of resources. The SEI will use existing software development datasets (e.g., ISBSG) and the software product complexity definition from the COCOMO® II software cost estimation model (Boehm et al., 2000) to define analogous software functions and the corresponding effort ranges. Basing the effort on software functions allows stakeholders to consider alternative solutions by adding or removing functions and pull comparable effort from various application types and domains.

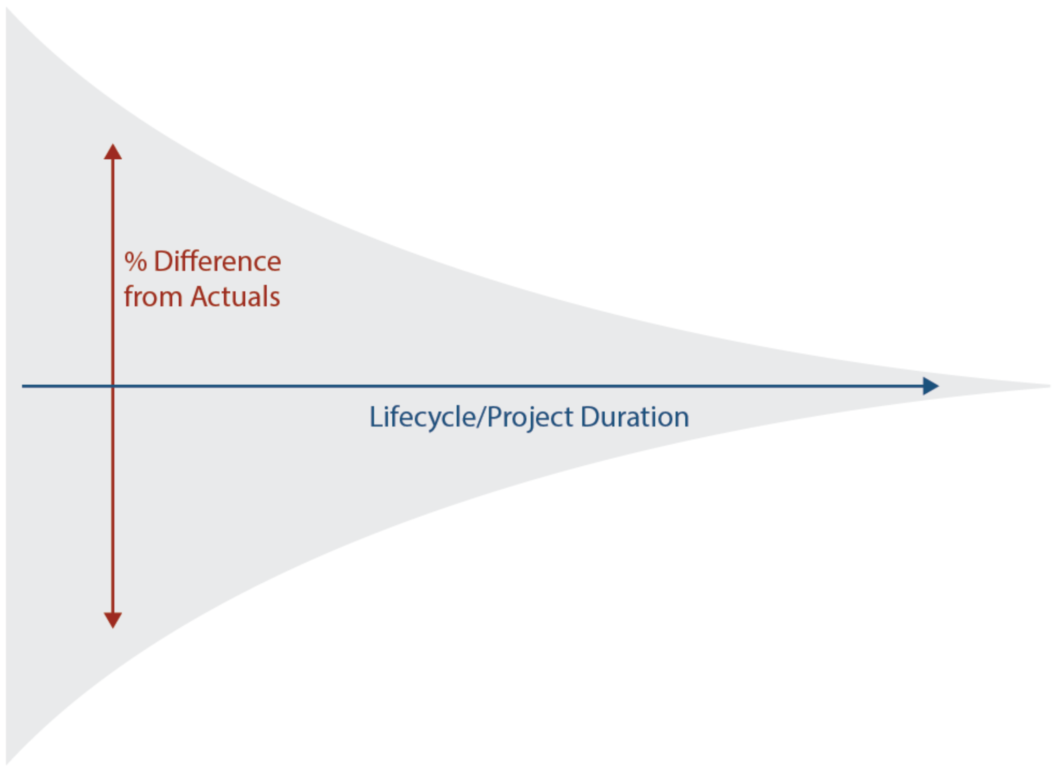


Figure 4. The Cone of Uncertainty

CaBSCE will not replace other software cost estimation methods. It will support early lifecycle estimates, when required capabilities are still being defined, and there is a substantial amount of architecture and implementation uncertainty. In 1981, Boehm developed the *"Cone of Uncertainty"* to demonstrate how uncertainty reduces over a project's lifecycle and leads to more accurate cost estimates. (See Figure 4 for a depiction of the Cone of Uncertainty, which was adapted from a graphic in Boehm's 1981 book.) CaBSCE provides an appropriate estimation method at the beginning of the lifecycle, while other estimation methods can be used as more information is learned and defined to refine the estimates. Additionally, other software cost estimation methods, (e.g., Function Points) can use the software functions grouping for more accurate estimates. Hira (2020) found that the relationship between Function Points and effort had significantly different trends based on the underlying software functions of the projects. Hence, CaBSCE will not only fill a gap in the current software cost estimation methods but will also provide software functions grouping to improve the accuracy of existing methods.

Acknowledgments

The author thanks her colleagues at the SEI—Eileen Wrubel, Bill Nichols, Julie Cohen, Chris Miller, Sandy Shrum, Barbara White, Jeff Mellon, Nick Testa, and Shannon Gallagher—and at the U.S. Air Force—Jackson Pontsler and Joseph Florio—for supporting and reviewing this study, and Glen Alleman (Niwot Ridge Consulting) for assisting with the example of relating capability to software functions.

Document Markings

References

Albrecht, A. J., & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions on Software Engineering*, *9*(6), 639–648. https://doi.org/10.1109/TSE.1983.235271

Alleman, G. (2015). *Capabilities based planning*. Herding Cats. https://herdingcats.typepad.com/my_weblog/2015/07/capabilities-based-planning.html

Alleman, G. (2019). *Capabilities based planning*. Herding Cats. https://herdingcats.typepad.com/my_weblog/2019/10/capabilities-based-planning.html

Alleman, G. (2020a). *Compendium of resources for capabilities based planning*. Herding Cats. https://herdingcats.typepad.com/my_weblog/2020/12/compendium-of-resoruces-for-capabilities-based-planning.html

Alleman, G. (2020b). *Eliciting capabilities*. Herding Cats. https://herdingcats.typepad.com/my_weblog/2020/03/eleciting-capabilities.html

Anastasios, P. (2014). *Capability-based planning with TOGAF and ArchiMate* [Master's thesis, Business Information Technology School of Management and Governance]. https://essay.utwente.nl/65421/1/Papazoglou_MA_MB.pdf

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M. … Thomas, D. (2001). *Principles behind the agile manifesto*. Agile Manifesto. https://agilemanifesto.org/principles.html

Bexfield, J., & Disbrow, L. (2004). Capabilities based planning: The road ahead. *Military Operations Research Society (MORS) Workshop.* Military Operations Research Society (MORS). https://apps.dtic.mil/dtic/tr/fulltext/u2/a443067.pdf

Biltgen, P. T. (2007). *A methodology for capability-based technology evaluation for system-of-systems* [Doctoral dissertation, Georgia Institute of Technology]. https://repository.gatech.edu/entities/publication/5ddd79af-ff47-4461-aeeb-33f04bf72dfb

Boehm, B. W. (1981). *Software engineering economics.* Prentice-Hall.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2000). *Software cost estimation with Cocomo II.* Prentice Hall.

Bradshaw, K. (2022). History repeats itself: A new revolution of factories... software! *Joint IT and Software Cost Forum.* https://www.dhs.gov/sites/default/files/2023-01/History_Repeats_Itself__A_New_Revolution_of_Factories..._Software_Remediated.pdf

Chairman of the Staff. (2009). *Capabilities-based assessment (CBA) user's guide (Version 3): Force structure, resources, and assessments directorate (JCS J-8)*. CreateSpace Independent Publishing Platform. https://acqnotes.com/wp-content/uploads/2014/09/Capabilities-Based-Assessment-CBA-Users-Guide-version-3.pdf

Chim, L., Nunes-Vaz, R., & Prandolini, R. (2010). Capability-based planning for Australia's national security. *Security Challenges, 6(3)*, 79–96. https://www.jstor.org/stable/26459800

Cohn, M. (2004). *User stories applied: For agile software development.* Addison-Wesley Professional.

Committee on Naval Analytical Capabilities and Improving Capabilities-Based Planning. (2005). *Naval analytical capabilities: Improving capabilities-based planning*. The National Academies Press. https://doi.org/10.17226/11455

Coonce, T., & Alleman, G. (2017). How should we estimate agile software development projects and what data do we need? *International Cost Estimating and Analysis Association (ICEAA) Professional Development & Training Workshop.* https://www.iceaaonline.com/wp-content/uploads/2017/07/AG01-Paper-Coonce-How-Should-we-Estimate-Agile-Software.pdf

Davis, P. K. (2002). *Analytic architecture for capabilities-based planning, mission-system analysis, and transformation*. RAND Corporation. https://studylib.net/doc/13665293/how-can-a-structured-representation-of-capabilities-help-...

Davis, P. K., Shaver, R. D., & Beck, J. (2008). *Portfolio-analysis methods for assessing capability options.* RAND Corporation. https://www.rand.org/pubs/monographs/MG662.html

De Lucia, A., & Qusef, A. (2010). Requirements engineering in agile software development. *Journal of Emerging Technologies in Web Intelligence, 2*(3), 212–220.

Defense Acquisition University. (2020a, October 2). *Cost estimation*. https://aaf.dau.edu/aaf/software/cost-estimation/

Defense Acquisition University. (2020b, January 23). *Software acquisition*. https://aaf.dau.edu/aaf/software/

Defense Innovation Board. (2019). *Software is never done: Refactoring the acquisition code for competitive advantage.* DoD. https://media.defense.gov/2019/May/01/2002126690/-1/-1/0/SWAP%20EXECUTIVE%20SUMMARY.PDF

Defense Science Board. (2009). *Report of the Defense Science Board Task Force on Department of Defense policies and procedures for acquisition of information technology.* DoD.

Desgagné, C. R. (2009). *Evolutionary acquisition – A complementary approach to capability based planning for the delivering of aerospace power* [Master of Defence Studies Research Project, Canadian Forces Command and Staff College]. https://www.cfc.forces.gc.ca/259/290/295/286/desgagne.pdf

DoD. (2021, March). *DoD enterprise DevSecOps fundamentals.* https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsFundamentals.pdf

GAO. (2020). *Cost estimating and assessment guide: Best practices for developing and managing program costs.* https://www.gao.gov/assets/gao-20-195g.pdf

Hales, D., & Chouinard, P. (2011). *Implementing capability based planning within the public safety and security sector: Lessons from the defence experience* [Technical Memorandum]. https://apps.dtic.mil/dtic/tr/fulltext/u2/a555463.pdf

Hanley, J. T., Jr., Fitzsimmons, M. F., Kurtz, J. H., Roark, L. M., Roske, V. P., Jr., & Cuda, D. L. (2006). *Improving integration of Department of Defense processes for capabilities development planning.* Institute for Defense Analyses. https://apps.dtic.mil/dtic/tr/fulltext/u2/a460395.pdf

Highsmith, J. A. (2002). *Agile software development ecosystems.* Addison-Wesley Professional.

Hira, A. (2020). *Calibrating COCOMO(R) II for functional size metrics* [Doctoral dissertation, University of Southern California]. https://digitallibrary.usc.edu/Share/hpp54fd5608060ex2w83q2het0p7dxeb

Iacobucci, J. V. (2012). *Rapid architecture alternative modeling (RAAM): A framework for capability-based analysis of system of systems architectures* [Doctoral dissertation, Georgia Institute of Technology]. https://repository.gatech.edu/server/api/core/bitstreams/e948f95b-591a-48a8-b47d-3a9cc6bb23ca/content

Idri, A., Amazal, F. A., & Abran, A. (2015). Analogy-based software development effort estimation: A systematic mapping and review. *Information and Software Technology, 58*, 206–230. https://doi.org/10.1016/j.infsof.2014.07.013

Jørgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software, 70*(1–2), 27–48.

Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communications of the ACM, 30*(5), 416–429.

Kossakowski, P. (2005). *Capabilities-based planning: A methodology for deciphering commander's intent*. McLean. http://www.dodccrp.org/events/10th_ICCRTS/CD/papers/319.pdf

Milani, M., & Rossi, M. (2020). Agile management versus budget control: Learn how to win both with Slick-Farm. *6th European Lean Educator Conference (ELEC 2019)*, 395–405. Springer.

Neaga, I., Henshaw, M., & Yue, Y. (2009). *The influence of the concept of capability-based management on the development of the systems engineering discipline*. Loughborough University. https://repository.lboro.ac.uk/articles/conference_contribution/The_influence_of_the_concept_of_capability-based_management_on_the_development_of_the_systems_engineering_discipline/9559580

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2020). *Operation of the software acquisition pathway* (DoD Instruction 5000.87). DoD. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF?ver=virAfQj4v_LgN1JxpB_dpA%3D%3D

The Open Group Architecture Forum. (2018). Capability-based planning. In *The TOGAF® Standard* (Ver. 9.2). https://pubs.opengroup.org/architecture/togaf91-doc/arch/chap32.html

Ozkaya, I., Kazman, R., & Klein, M. (2008). *Understanding the relationship of cost, benefit, and architecture.* Carnegie Mellon University, Software Engineering Institute.

Planeaux, J. B. (2003). *Beyond the task force conops: The path to a capabilities-based modernization framework for the Air Force* (Research Report). Air War College, Air University, Maxwell Air Force Base. https://apps.dtic.mil/dtic/tr/fulltext/u2/a424623.pdf

Raza, S., & Waheed, U. (2018). Managing change in agile software development a comparative study. *IEEE 21st International Multi0Topic Conference (INMIC).* IEEE.

Rosa, W., Clark, B. K., Madachy, R., & Boehm, B. W. (2021). Empirical effort and schedule estimation models for agile processes in the US DoD. *IEEE Transactions on Software Engineering*, *48*(8), 3117–3130.

Shah, T., & Patel, S. V. (2014). A review of requirement engineering issues and challenges in various software development methods. *International Journal of Computer Applications, 99*(15), 36–45.

Shepperd, M., & Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on Software Engineering, 23*(11), 736–743.

Steele, J. A. (2021). Capability-based planning and the Royal Canadian Air Force. In *RCAF Defence Economics* (pp. 77–131). Canadian Forces Aerospace Warfare Centre. https://cradpdf.drdc-rddc.gc.ca/PDFS/unc357/p812921_A1b.pdf

Taliaferro, A. C., Gonzalez, L. M., Tillman, M., Ghosh, P., Clarke, P., & Hinkle, W. (2019). *Defense governance and management: Improving the defense management capabilities of foreign defense institutions: A guide to capability-based planning (CBP).* Institute for Defense Analyses. https://www.ida.org/research-and-publications/publications/all/d/de/defense-governance-and-management_improving-the-defense-management-capabilities-of-foreign

Technical Cooperation Program. (2004). *Guide to capability-based planning*. DoD. https://www.hsdl.org/?view&did=461818

Titus, N. (2004, March). *Air Force CONOPS & capabilities based planning.* U.S. Air Force. https://www.hsdl.org/?view&did=453278

Van Hai, V., Nhung, H. L., Prokopova, Z., Silhavy, R., & Silhavy, P. (2022). Toward improving the efficiency of software development effort estimation via clustering analysis. *IEEE Access*, *10*, 83249–83264.

Walker, S. K. (2005). *Capabilities-based planning – How it is intended to work and challenges to its successful implementation.* U.S. Army War College. https://apps.dtic.mil/sti/pdfs/ADA434864.pdf

Webb, M. (2008). Capabilities-based engineering analysis (CBEA). In A. Minai, D. Braha, & Y. Bar-Yam (Eds.), *Unifying Themes in Complex Systems: Proceedings of the Sixth International Conference on Complex Systems* (pp. 540–547). https://doi.org/10.1007/978-3-540-85081-6_67