ABSTRACT

Title of Dissertation: ANALYSIS OF THE LIFE-CYCLE COST AND CAPABILITY TRADEOFFS ASSOCIATED WITH THE PROCUREMENT AND SUSTAINMENT OF OPEN SYSTEMS

Shao-Peng Chen, Doctor of Philosophy, 2024

Dissertation directed by: Professor Peter A. Sandborn
Department of Mechanical Engineering

System openness refers to the extent to which system components can be independently integrated, removed, managed, or replaced without adversely impacting the system. Openness (of a system and/or architecture), though intuitively understood, remains difficult to quantify in terms of its value for safety-, mission-, and infrastructure-critical systems. Examples of these critical systems include: aircraft, rail, industrial controls, power generation, and defense systems – all of these systems are characterized by large procurement costs, large life-cycle sustainment costs and very long support lives (e.g., it is not uncommon for these systems to be supported for 30 or more years).

Generally, it is taken for granted that the use of open systems decreases the total life-cycle cost of a system. Leveraging existing open technology, including commercial-off-the-shelf (COTS) components, avoids many costs associated with designing custom components, and reduces the time required for development and eventually refresh of a system. The use of open systems helps mitigate the effects of obsolescence, lengthens the system's support life, and allows for the incremental insertion of new technologies. Component design reuse also

eliminates redundant components, thus reducing logistical costs.

However, building systems from open standards and commercially available components often relies on the use of generalized technology containing unnecessary additional functionality, which increases the system's complexity and adds new failure paths and additional qualification overhead. In other cases, it may be necessary to modify COTS components to meet performance requirements, thereby adding costs. In addition, the enterprise that manages the system often has no control over the supply-chain for COTS components, which adds supply disruption risk and introduces the risk compromised.

Previous efforts to establish the value of openness have relied on highly qualitative analyses, with the results often articulated as intangible "openness scores". Such approaches do not provide sufficient information to make a business case or understand the conditions under which life-cycle cost avoidance can be maximized (or whether there even is a cost avoidance).

This dissertation is focused on creating a general model for quantifying the relationship between system openness and life-cycle cost that can be used to optimize system openness strategies for critical systems, an outcome that could significantly reduce system sustainment costs. This work is composed of the following tasks: 1) Mining of public-source materials to solidify what is known and believed about the relationship between open-systems attributes and life-cycle costs. 2) Development of a multivariate model and associated simulation that quantifies the relationship between openness and life-cycle cost for systems composed of hardware and software. 3) A case study of the life-cycle cost difference between two implementations of the same system with differing levels of openness (the US Navy A-RCI sonar system on Los Angeles and Ohio class submarines is the case study system for this dissertation). 4) Generalization of the understanding of the relationship between system life-cycle cost and openness is achieved through a generic analytic model. This model efficiently

estimates the relationship between relative life-cycle cost and system openness, considering relevant parameters. It enables the determination of optimum system openness without the need for running a detailed simulation.

ANALYSIS OF THE LIFE-CYCLE COST AND CAPABILITY TRADEOFFS ASSOCIATED WITH THE PROCUREMENT AND SUSTAINMENT OF OPEN SYSTEMS

by

Shao-Peng Chen

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:
Professor Peter A. Sandborn, Chair
Professor Bilal M. Ayyub
Professor Katrina Groth
Professor Jeffrey Herrmann
Professor William Lucyshyn

Dedication

I dedicate this dissertation to my father, Chao-En Chen, who was my hero.

Acknowledgement

I would like to express my deepest gratitude to my academic advisor, Professor Peter A. Sandborn, for giving me the research opportunity and introducing me to the world of life-cycle cost analysis. His guidance and support throughout my research have been invaluable. He allowed me the freedom to pursue my ideas and provided help whenever I faced difficulties, whether in research or my career. I am truly grateful to have him as my advisor and have learned so much from him.

I also wish to extend my thanks to my defense committee members, Professor Bilal M. Ayyub, Professor Katrina Groth, Professor Jeffrey Herrmann, and Professor William Lucyshyn. Their expertise in reliability engineering and their insightful comments and feedback have significantly enriched my dissertation.

I am deeply thankful to my family, the Chen family. My parents, Chao-En and Pei-Hua, and my sisters, Shao-Yi and Shao-Chien, have always been my role models. I would not be where I am today without their unwavering support and encouragement.

I would also like to thank my friends from the University of Maryland and the DMV volleyball community for providing a supportive environment for my mental well-being.

Finally, I want to express my heartfelt appreciation to my life partner, Chia-Yun Chang, for believing in me from the beginning and for engaging in countless discussions about my research throughout my student life. She is the most supportive and understanding person I know, and her unwavering belief in me has been a constant source of strength.

Table of Content

List of Tables

List of Figures

List of Abbreviations

A-RCI = Acoustic Rapid COTS Insertion

COCOMO = COnstructive COst MOdel

COTS = Commercial Off the Shelf

CBS = COTS-Based System

DoD = Department of Defense

HW = Hardware

MoD = Ministry of Defense

MOSA = Modular Open Systems Approach

NRE = Non-Recurring Engineering

O&MN = Operations and Maintenance, Navy

O&S = Operation and Support

OAAM = Open Architecture Assessment Model

OAAT = Open Architecture Assessment Tool

OAET = Open Architecture Enterprise Team

OPN = Other Procurement, Navy

OSA = Open Systems Approach

OSJTF = Open Systems Joint Task Force

PART = Program Assessment and Rating Tool

RDT&E = Research, Development, Test, and Evaluation

SCN = Shipbuilding and Conversion, Navy

SOSA = System of Systems Approach

SW = Software

TTF = Time to Failure

Chapter 1 Introduction

Manufacturers and sustainers of critical systems face many long-term budgetary challenges. Critical systems, such as aircraft, rail, industrial controls, power generation, defense and communications infrastructure, are prohibitively expensive to procure and sustain over their long-life cycles (measured in decades). For systems where state-of-the-art performance is required throughout their life cycle, the budgetary challenge is more severe due to system refresh and upgrade. In order to maintain the capability required to remain competitive or effective in accomplishing their intended purpose, refresh and upgrade are constantly required so that obsolete and outdated components can be replaced with supportable and state-of-the-art components. As a result, future spending needs to be as efficient as possible, which will require many hard decisions and will force the reengineering of processes.

Critical systems have traditionally been developed using acquired proprietary systems and interfaces, which make it challenging to modernize and reduces opportunities for competition. For example, the U.S. Air Force's program to upgrade the B-2 bomber's communications, networking, and defensive management systems will cost over $2 billion, since the prime contractor owns all the necessary proprietary technical data and software. Because of the "closed" nature of the system, competing this effort was not a viable economic option, (GAO, 2014). In addition, proprietary systems and interfaces also made the system costly to refresh since the customized components had high development and production cost and the proprietary interfaces also made it difficult to upgrade components individually. That is, when a component was changed in the architecture, many other components needed to be changed as well.

A variety of strategies are being explored, or reemphasized, to increase the efficiency of acquisition processes. One way for the defense community (and critical systems in general) to

minimize the cost and time needed to modify or upgrade systems is by using an Open Systems Approach (OSA) for system design and development. When used appropriately, OSA provides a degree of flexibility, enabling the integration of rapidly changing technologies. However, as with all approaches, there are costs as well as benefits. This dissertation explores a business case methodology to assess the application-specific cost effectiveness of OSA.

1.1 Open Systems Approach

To delve into the open system approach, it is essential to establish the concept of an open system. Conceptually, a system is considered to be more "open" when it is more adaptive and flexible to system change, which may be due to the requirement of either performance upgrade or functionality extension. The DoD Open Systems Joint Task Force defined an open system as:

"A system that implements sufficient open specifications for interfaces, services, and supporting formats to enable properly engineered components to be utilized across a wide range of systems with minimal changes, to interoperate with other components on local and remote systems, and to interact with users in a style that facilitates portability."

Generally, an open system is characterized by a system architecture built on widely supported standard interface, facilitating the efficient use of the commercial products throughout during system development and upgrade (Kowalski et al., 1998; Dargan, 2005). It has been widely believed that open systems contribute to life-cycle cost reduction and are able to adopt to more frequent capability upgrade since it can efficiently fulfill a "plug-and-play" component upgrade process.

Open systems approaches have been widely used in both defense and industry area. The U.S. Department of Defense (DoD) established a program in 1994 to promote the use of open-systems approaches from the top-down. In fact, the acquisition strategy for a given system must

identify where, why, and how modular open systems will be used, (DoD, 2015). The DoD's Open Systems Approach, now referred to as Modular Open Systems Approach (MOSA), promotes the use of modular design to encourage companies to improve and manufacture technologies that are interoperable with the DoD's current system. Formally, MOSA is defined as "a technical and business strategy for designing an affordable and adaptable system" (Defense Standardization Program, 2002).

During the 2010s, the Department of Defense (DOD) mandated the integration of Open Systems Architecture (OSA) principles and practices into all acquisition activities. This directive emphasized the use of existing open standards and market-sourced hardware/software components. Additionally, in 2019, a tri-service memorandum mandated the inclusion of Modular Open Systems Approach (MOSA) approved standards in all future weapon system requirements, programming, and development activities to the fullest extent possible (Office Secretaries of the Navy, Army, and Air Force, 2019).

Throughout this period, the Navy, Army, and Air Force each initiated programs to adopt open systems, actively developing, demonstrating, and validating open standards through collaborative partnerships with industry and academia. These programs include the Army's CCDC (Combat Capabilities Development Command), the Navy's HOST (Hardware Open Systems Technology), and the Air Force's OMS (Open Mission Systems). In 2017, The Open Group established the SOSA (Sensor Open System Architecture) Consortium, integrating contributions from US DOD services, industry, academia, and other government organizations (Hosking, 2020). Focused on creating a common open standard/architecture, the SOSA consortium aims to enable the reuse of hardware and software modules across various sensor types and systems (Shepherd & Wills, 2018).

The United Kingdom's Ministry of Defense (MoD) also has a similar strategy for the use of

open-systems architecture; they refer to it as the System of Systems Approach (SOSA) Open Systems Strategy. The SOSA Open Systems Strategy describes the open-systems vision and the roadmap for achieving the required level of openness across the defense enterprise. The MoD's policy provides high-level guidance and states that an open-systems approach should be adopted to realize benefits that include: ease of interoperability, ease of modification, improved integration, improved opportunities for competition and innovation, and improve obsolescence management, (UK MoD, 2013). The UK, along with several other countries (Norway, Germany, and Sweden) have also adopted programs to upgrade their submarines' Combat Management Systems using an open systems approach and COTS (Commercial Off the Shelf) components (Peruzzi, 2019). Finally, although the Australian Defence Forces do not mandate the use of open system architectures, a 2016 Defence White Paper identified the opportunities offered using open-system design concepts (Dunn et al., 2018).

In conclusion, an open systems approach (OSA) is a series of business or engineering strategies that can be used to make a system more open, and the strategies comprise the aspects of architecture design, component selection and program management. Since any strategy that could make a system more open can be considered as part of an open-systems approach, it is difficult to create a simple statement to completely describe what an open-systems approach should be. Table 1.1 lists some of the abilities that are considered to be an open-systems approach.

Table 1.1 Open Systems Architecture Assessments

System aspect	Detail description
Architecture design	• Using modular design • Using commercial standards • Ability to easily respond to evolving requirements (technology refresh) • Ability to evaluate the appropriateness of using open standards for key interfaces • Using the same data model so subsystems can exchange information and appropriately utilize each other's functional capabilities (interoperability)

	• Creating an extensible architecture • Minimizing the scope of the test after adding new component to the architecture
Component selection	• Using components from the commercial market (COTS) • Using proprietary components that conform to standards • Establishing testing mechanisms to verify and validate that components conform to standards • Reducing number of components that are vendor specific
Program management	• Relying on the commercial market, allowing the marketplace to guide technology development and direction • Creating an environment for third parties to compete for functional modules or components freely, not rely only on one source (avoiding vendor lock) • Ability to have independent sources competing, participating in system upgrade • Establishing data management strategies that ensure Government exercise of intellectual property rights. • The ability to reuse component from other programs

1.2 Why Open System

The increasing evolving speed of technology which magnified the advantage of OSA, leading to their wide acceptance by business and industry. Historically, critical functionality in complex electronic systems was provided by custom-made components and custom proprietary architectures, requiring long development times and high development costs. However, recent technological advancements have allowed for the increased generalizability of both hardware and software (and system architecture); now components can be designed once, and then used in many different applications. These advancements have increased the viability of using OSA in general, and a Modular Open Systems Approach (MOSA) in particular (Abbott et al., 2008).

An open-systems approach, when used in conjunction with a modular architecture, reuse, and/or the harnessing of existing technologies (COTS or proprietary), is commonly associated with cost avoidances arising from more efficient design, increased competition among

suppliers, more efficient innovation and technology insertion (faster, cheaper design evolution), and the modularization of qualification. The use of OSA or MOSA helps mitigate the effects of obsolescence, lengthens the system's support life, allows for the incremental insertion of new technologies (OSJTF, 2004; Boudreau, 2006), and evolving functionality. Leveraging existing open technology, including COTS components, avoids many costs associated with designing custom systems, and reduces the time required for development or refresh of a system (Logan, 2004). The use of well-defined standards promotes smooth interfacing both within and between systems, while the proliferation of common component types fosters competition between suppliers. Component design reuse (within and between systems) eliminates redundant components, thus reducing logistical costs.

1.3 Open Systems Challenges

While the defense community supports implementing OSA whenever possible, there are numerous reasons to be cautious since business and engineering-tradeoffs must be made, possibly changing the incentive structure and reducing the system effectiveness. First, if there are no standards for a new product, then closed system architecture may be best until standards are created (Firesmith, 2015). Second, a poorly designed modularized architecture may be too costly to re-architect and it's better to stick with the existing system. Third, there is only one qualified vendor to provide the service, making opening the system costly without benefit. These drawbacks challenge implementing MOSA, but neither the open system nor closed systems side of the argument has successfully provided quantitative analysis to support their position.

There are costs associated with openness that also should be considered. Building a subsystem from open standards and commercially available components often relies on the use of generalized technology with unnecessary and costly additional functionality, increasing the

cost, complexity, and effective failure rates (Hanratty et al., 2002; Bass et al., 2008). In other cases, it may be necessary to modify COTS components to meet performance requirements (Wright et al., 1997; Jensen and Petersen, 1982), thereby adding costs. In addition, the enterprise that manages the system likely has no control over the supply chains for COTS components, which tend to be more volatile than proprietary ones (Lewis et al., 2000). This makes it desirable to refresh open systems designs more frequently (Clark and Clark, 2007; Abts, 2002), which leads to an increase in the number of fielded configurations, which complicates logistics, resulting in more expense.

Therefore, determining if, and to what extent, openness should be pursued remains challenging in the absence of a quantitative model that elucidates the relationship between the degree of system openness and the system's life-cycle cost.

1.4 Existing Work and Research Gaps

Several previous efforts have addressed the measurement of system openness. These include the MOSA Program Assessment and Rating Tool (PART), developed by the U.S. Navy's Open Systems Joint Task Force (OSJTF). PART consists of a series of questions, divided into business indicators and technical indicators, that assess the extent to which a particular principle or practice is implemented. PART can be used to measure a system's openness, but it is subjective and qualitative, and the results depend on the optimism with which the survey is completed (OSJTF, 2004).

The Naval Open Architecture Enterprise Team (OAET) developed the Open Architecture Assessment Model (OAAM) to "define, measure, and illustrate the relative levels of openness" (NOAET, 2009). Like PART, OAAM measures the openness of a system using both business and technical characteristics, which are combined to produce an overall openness characterization. While OAAM was relatively simple, it lacked resolution, so OAET created

the Open Architecture Assessment Tool (OAAT) to expand and build upon OAAM (NOAET, 2009). OAAT uses a questionnaire that includes PART to assess system openness (NOAET, 2009).

In 2011 and 2012 several organizations, including the U.S. Air Force Research Laboratory's RYM subgroup collaborated to develop a set of metrics to evaluate the openness of an architecture. This effort focused on selecting metrics that were broad enough to assess a general case, and quantifiable, so that the measurement would be repeatable. The result of this effort, called the MOSA Metrics Calculator (MOSA, 2012), improves upon the PART questionnaire by ensuring that all metrics are quantifiable.

One common attribute of the PART, OAAT, and the MOSA Metrics Calculator is that they don't account for the cost associated with an open systems approach and are unable to measure the value of the benefits obtained. They cannot be used to make a business case for the use of openness, i.e., they explicitly assume that increased openness is always beneficial, but is it?

Another approach to measuring openness comes from PMH Systems and the University of Southampton. This work uses an easily quantifiable metric, the fraction of interfaces that use open standards, and a stochastic model to estimate the decrease in cost and development time associated with increasing openness (Henderson, 2009). However, the model implicitly relies on the assumption that increased openness is always beneficial. Additionally, the metric developed cannot resolve different levels of openness and most importantly only addresses the design phase, ignoring significant costs and avoidances that occur later in the system's life cycle.

In the software industry, the concept of COTS has been widely discussed since late 1990s and the life-cycle cost models of COTS-based systems have been proposed by several researchers. COCOMO (Constructive Cost Model) is an empirical parametric model that can

be used to estimate the time, effort, and cost associated with developing a software system. Based on COCOMO, CICC (COTS Integration Cost Calculator) and COCOTS (Constructive COTS Cost Model) were developed to estimate the costs incurred to integrate a COTS software component into a larger system during the design phase (Abts and Boehm, 1997; Abts et al., 2000). In addition, several cost models associated with COTS systems were built, drawing inspiration from COCOMO and COCOTS. Minkiewicz (2001) introduced an activity-based cost algorithm identifying COTS software component costs throughout the life cycle. Yang (2005) incorporated COTS reuse and assessed the benefits of accelerated time-to-market through COTS component implementation. Erdogmus et al. (2000) proposed a framework emphasizing the iterative development nature of software COTS-based systems. Leffet et al. (2011) built a cost estimation model considering various methods of COTS-based development. However, these models predominantly focus on the impact of COTS software implementations, lacking consideration for the cost effects of general system openness.

While increased openness doesn't necessarily assure cost-effectiveness, some studies have concentrated on constructing evaluation frameworks to determine the optimal level of system openness. Lebron Jr et al. (2000) devised a risk-based COTS assessment model aiding military program managers in COTS technology decisions by evaluating the relative risks of each alternative. Colombi et al. (2015) formulated a framework based on multi-objective decision analysis to pinpoint key interfaces for open standard implementation, aiming to maximize the value of open standards. Despite offering decision-making criteria for COTS components and open standards, these studies relied heavily on subjective scoring and qualitative analysis. While many studies on system openness consider life-cycle cost as a decision-making factor, few delve into illustrating how life-cycle costs can be estimated based on system openness. A notable exception is Schramm (2013), who developed a discrete-event simulator to estimate life-cycle costs with given system openness, demonstrating through simulated results that

increased openness doesn't always guarantee cost avoidance.

In conclusion, past studies associated with system openness can be classified into three groups. The first group relied on a highly qualitative analysis of a system, with the results often given as an intangible "openness score" used to determine which of multiple system implementations is more open. The second group focus on developing quantitative COTS-based system life-cycle cost models for software only, where the cost effect of other openness factors such as open standard implementation was not considered. The last group provided evaluation frameworks to determine the optimum extent of implementing COTS components and open standards, these are qualitative analysis and were highly dependent on subjective scoring. Such approaches do not provide enough information to make a business case or understand the conditions under which life-cycle cost avoidance can be maximized (or whether there even is cost avoidance).

The fundamental research gap is that none of the existing studies have built a comprehensive quantitative relationship between system openness and its associated benefit/cost (the only existing quantitative work is for software systems only, e.g., the COCOMO, CICC (COTS Integration Cost Calculator) and COCOTS proposed by Abts et al (1997).). This dissertation aims to quantify the relationship between system openness and life-cycle cost by developing a model to enable answering the following key questions on an application-specific basis:

- What are the costs avoided and added due to openness?
- What are the variables that should be considered in assessing the cost impact associated with system openness?
- What level of openness provides the best value (largest cost avoidance) to the customer?
- What "rules of thumb" regarding openness can be generalized based on the

developed model?

1.5 Dissertation Scope

According to the *Systems Engineering Guidebook* (Office of the Deputy Director for Engineering & Office of the Under Secretary of Defense for Research and Engineering, 2021), the systems engineering process (SEP) is a framework that allows programs to structure and conduct technical efforts efficiently and effectively to deliver capabilities that satisfy validated operational needs. This includes methodical and disciplined approaches for the specification, design, development, realization, technical management, operations, and retirement of a system. Figure 1.1 presents a general overview of the SEP, highlighting that the technical processes involve top-down decomposition and bottom-up realization to iteratively develop and validate system solutions. Concurrently, the technical management processes monitor and control technical activities to ensure program success. Together, these processes enable the delivery of operational capabilities while mitigating technical and programmatic risks.

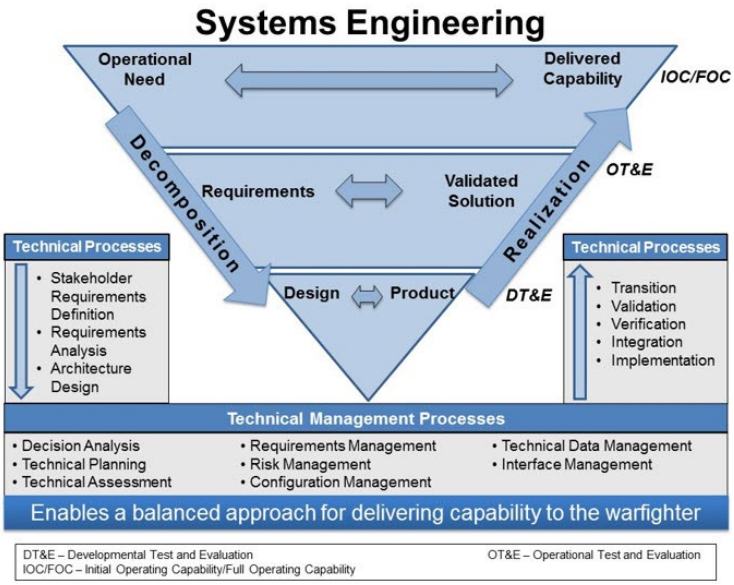


Figure 1.1 The Systems Engineering Processes in Systems Engineering Guidebook (Office of the Deputy Director for Engineering & Office of the Under Secretary of Defense for Research and Engineering, 2021).

The Modular Open Systems Approach (MOSA) and Commercial-Off-the-Shelf (COTS) solutions were discussed in the *Systems Engineering Guidebook* as key system-level and design considerations for acquisition and design strategy. They are involved in several processes shown in Figure 1.1, such as architecture design, decision analysis, and technical planning. The benefits of MOSA and COTS include increased interoperability, reduced development time, faster technology refresh and evolutionary upgrades, and potential cost savings or cost avoidance.

Although the open systems approach can be adopted and impact many aspects of the systems engineering process, and offer various benefits in interoperability, reuse, and life-cycle cost, this dissertation focuses specifically on the selection of components and interfaces. It does not consider how modular and loosely coupled the system architecture should be, the selection of vendors to provide innovation, or the reuse of system components. Instead, the focus is on determining whether to use open or closed components/interfaces based on the economic influence of the open systems approach, specifically life-cycle cost analysis. It is assumed that other aspects of the open systems approach, such as architecture modularity, are either already defined or need to be analyzed in other processes.

1.6 A Working Definition of Openness

A common definition of the Open System Approach in various research studies and applications revolves around "using open interfaces supported by commercial and non-developmental components." Therefore, within the context of this dissertation, a working definition of openness is proposed, focusing on the selection of the type of components and interfaces. The degree of openness in a system is characterized by its adoption of open components, such as Commercial Off-The-Shelf (COTS), and open interfaces, exemplified by open standards. The selection of the components and interfaces can serve as measurable

decision variables, forming the basis for a quantitative analysis of the costs associated with system openness.

Besides, modular architecture is also acknowledged as a crucial element in Open Systems Architecture (OSA) and several studies have delved into the modularity of system architecture, showing that modular architecture can be cost beneficial to the system sustainment as well (Davendralingam et al., 2019). However, due to the predefined scope of this dissertation, the focus is on the selection of components and interfaces within an already established architecture. This dissertation assumes that the architecture, or network, is in place, leaving the subsequent task of determining the optimal selection of components and interfaces within this established framework.

According to this working definition, the higher the fraction of COTS components and the more extensive use of open standards, the greater the degree of openness in a system. However, the fraction of open components/interfaces is not the sole factor influencing the system life-cycle cost. The topology, i.e., where the COTS components and the open standards located in the system architecture, also play a crucial role in impacting the cost/benefit of an open system. For instance, open standards can either increase or decrease costs depending on the interfaces to which they are applied. Key interfaces linked to components necessitating more frequent refresh cycles, can benefit from reduced modification costs and shorter integration timelines throughout the system's lifetime. Conversely, applying open standard to non-key interfaces may consume resources in pursuing an open systems approach when it is unnecessary (Colombi et al., 2015). Thus, when constructing a model for the relationship between life-cycle cost and openness, it is imperative to consider both the quantity and placement of open components/interfaces to formulate a comprehensive life-cycle cost model.

1.7 Problem Formulation

OSA is believed to be one of the key elements to overcoming the budgetary challenges that critical systems are currently facing. However, to realize the value of OSA, one must be able to determine the optimum application-specific degree of openness necessary to maximize the cost avoidance, in other words, obtaining the minimal life-cycle cost. This dissertation seeks to create a procedure for minimizing system life-cycle costs by considering the degree of openness as the decision variable. This involves two key components: 1) Formulating the objective function (f) that represents the decision variable (x), wherein the life-cycle cost is expressed as a function of the working defined openness in conjunction with other parameters related to life-cycle costs. 2) Implementing an approach to evaluate the objective function, which encompasses calculations, simulations, or other analytical techniques to comprehend the influence of changes in the degree of openness on the overall life-cycle cost.

This dissertation proposes a relative life-cycle cost model[1] as the objective function (f) instead of an absolute life-cycle cost model. A relative-cost model is proposed because it is effectively impossible to consider every cost throughout a system's life, and many of the life-cycle costs are irrelevant to the question of system openness, i.e., they are not impacted by the degree of system openness. To quantify the relationship between openness and life-cycle cost, a qualitative analysis is first performed to evaluate the dependency between system openness, the contributing factors, and life-cycle cost elements. An influence diagram is shown in Figure 1.2 that identifies the elements associated with system openness and their effects on various life-cycle cost components. The arrows show the influence direction or dependency. The shapes of the nodes indicate their corresponding types: value (round-corner box) and decision (rectangle).

[1] In the rest of this dissertation, we use the term life-cycle cost is going to refer the "relative" life-cycle cost.

Building the influence diagram involves identifying the connections and dependencies between the decision variables and life-cycle cost elements. We first identified the starting points, i.e., decision variables, which include the selection of the components (COTS or proprietary), interface (open or proprietary), and refresh strategy. The end points are the life-cycle cost elements, which include development, production, O&S (operations and support), refresh, and capability costs. We then analyze the impact of each decision variable and how these impacts affect the cost elements. For example, the selection of COTS or proprietary components can directly affect the individual component development costs and component procurement costs, ultimately determining the development and production costs of the life cycle. This relationship is shown in the influence diagram by placing arrows between boxes to indicate dependency. Through this analysis, we can establish the initial relationships between the decision variables and life-cycle cost elements.

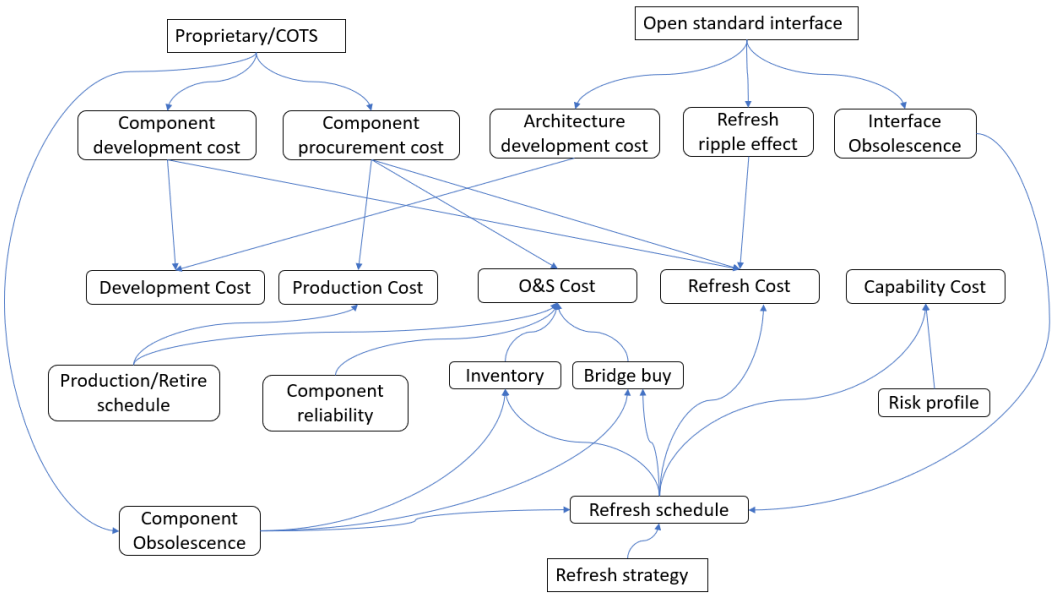


Figure 1.2 Influence diagram of life-cycle cost associated with system openness.

Based on the influence diagram in Figure 1.2, a conceptual quantitative model of life-cycle cost is formulated. The decision-making problem of system openness to minimize life-cycle cost can be written as,

$$\min_{g,s} Z = f(g,s,p)$$

$$s.t.\ \ q_k(g,s) \leq 0;\ \ k = 1,2,\dots K \qquad (1.1)$$

$$g \in G$$
$$s \in S$$

In Equation (1.1), the relative life-cycle cost function $f(g,s,p)$ includes three factors which are system architecture g, refresh strategy s and the parameter set p. The first two factors are the decision variables of the life-cycle cost function while the parameter set p is predetermined. $q_k(.)$ is the constraints of system architecture g and refresh strategy s. g and s should be within each domain set, G and S. The objective function Z is the relative life-cycle cost given the decision variables, g and s. The optimal objective function value Z^* should be in the range between $[0, \infty)$.

The system architecture, denoted as g, is illustrated in the form of a network. The practice of mapping a system architecture into a network has been widely utilized in system engineering studies (Tamaskar et al., 2011; Paul, 1998). In an architecture network, individual nodes correspond to components, and the links between nodes depict interfaces that define the interactions between components. In addition to its representation of the system architecture network, g also captures the system openness information. Each node and link can be assigned specific designations as open or closed components and interfaces, thereby indicating the degree of openness and location of the open/closed component/interface within the architecture. This method proves to be not only straightforward in representing the system architecture but also effective for later simulating life-cycle costs.

s is the refresh strategy. An optimum life-cycle cost is a result of a joint decision of system openness and refresh strategy. Therefore, besides system openness, refresh strategy is also a critical factor. S includes a series of different refresh strategies that could be implemented impacting different types of components in the system. Each individual refresh strategy

includes information on the interval of development of a new architecture baseline and the interval of refresh delivery to fielded systems. In certain scenarios, the choice of the refresh strategy may be independent of the life-cycle cost, meaning a specific strategy is enforced due to capability requirements. In such cases, as illustrated in Chapters 5 and 6, the refresh strategy becomes a predetermined parameter and is included in the parameter set instead of being treated as a decision variable.

p denotes the parameter set, which encompasses factors contributing to life-cycle cost independently of system openness but may collectively influence life-cycle cost when considered alongside system openness. These parameters are presumed to remain constant in the life-cycle cost model and can be either deterministic or random. The elements included in parameter set p comprise costs associated with components and interfaces, support lives of components and standards, production/retirement schedules of systems, risk profiles, and other relevant factors.

Up to this stage, the inputs g, s, and p associated with life-cycle costs have been introduced. The subsequent challenge involves determining the most appropriate approach for formulating and computing life-cycle costs based on these inputs. The traditional mathematical approach, which entails establishing analytical expressions for objective and constraint functions, may not be the most suitable starting point given the intricate nature of the system. There are two primary reasons for this:

1. In the context of a complex architecture with tens to hundreds of different component and interface types, devising simple analytical forms to represent each component/interface can be challenging.
2. Throughout the life cycle of a complex system, numerous interactions among components/interfaces occur, particularly during obsolescence and refresh, and are

difficult to be formulated as analytic functions. For instance, the obsolescence of one component can influence whether other components connected through interfaces need to be refreshed. These interactions directly impact the life-cycle cost, and capturing them adequately through analytical formulation proves challenging.

To formulate the life-cycle cost with random variables and effectively capture complex interactions between them while seeking the optimal openness, a discrete-event simulation serves as a more suitable starting point. Simulation-based optimization[2], as outlined by Deng (2007), will be employed. The objective function Z, as presented in equation (1.1), is approximated by computing the average of total simulation results, as illustrated in equation (1.2), where n denotes the total number of trials. Addressing uncertainty in the parameter set p involves repeated random sampling. In each trial, discrete-event simulation is employed to simulate event occurrences and the system's state changes over time. Based on these simulated events and states, the relative life-cycle cost is calculated, resulting in $\mathbb{Z}$ representing the relative life-cycle cost difference for each trial. The objective function Z is then expressed as the average of these $\mathbb{Z}$ values.

$$\min_{g,s} Z(g,s,p) = \min_{g,s} \mathbb{E}[\mathbb{Z}(g,s,p)] \approx \min_{g,s} \frac{1}{n}\sum_{i=1}^{n} \mathbb{Z}(g,s,p_i) \tag{1.2}$$

where n is the number of total trials.

1.8 Dissertation Objective and Research Tasks

In this dissertation, a simulation model is initially developed to simulate life-cycle costs based on the established relationship between cost and openness. Subsequently, a case study is conducted on the US Department of Defense's A-RCI sonar system, recognized as a successful

[2] Simulation-based optimization, which integrates simulation and optimization techniques, has been successfully applied to solve complex real-world problems in various domains such as finance, business management, and engineering design. In this dissertation, the focus of the simulation-based optimization concept is on formulating the problem rather than emphasizing the optimization technique.

case of Open Systems Architecture (OSA). The resulting model serves as a virtual environment for extrapolating the quantitative relationship between openness and life-cycle cost. Building upon the simulation model, a general analytic model is proposed, incorporating additional assumptions. The objective of this analytic model is to efficiently estimate the relationship between life-cycle cost and system openness without the need for running the entire simulation.

The research tasks in this dissertation include the following:

- Task 1: Develop a model that can map the inputs including system architecture, refresh strategy and other system parameters into life-cycle costs. System openness may have influence on some cost factors, e.g., less development cost for COTS, and other system factors such as the propagation of refresh effects. The relationship between system openness and these factors and how they affect life-cycle cost should be realized and modeled. The system architecture input should also capture the components and their interaction with each other in the system. The types of components and the interfaces that represent system openness should also be included.
- Task 2: Implement the model from Task 1 to simulate the cost difference between two system scenarios by with varied decision variables, i.e., system openness g and refresh strategy s. The model is a stochastic discrete-event simulation that can determine life-cycle cost by generating the events and accumulating the corresponding discounted cost.
- Task 3: A-RCI sonar system case study
    - Task 3a: Identify and compile A-RCI input data. The information characterizing A-RCI includes system architecture, production/retirement schedule, and refresh strategy, which are collected and abstracted for the inputs for the model.
    - Task 3b: Calibrate the model coefficients based on actual cost data for the A-RCI. Some of the model coefficients are unavailable and therefore they must be reverse engineered from existing life-cycle cost data available for the A-RCI program.

- Task 3c: Analyze the simulation result based on the A-RCI case study. This task compares the life-cycle cost difference between the actual A-RCI with other scenarios with varied the decision variables, system architecture g and refresh strategy s. Sensitivity analysis of selected parameters such as fleet size and risk profile are also conducted.

◆ Task 4: Formalize the quantitative correlation between system openness and life-cycle cost. Building upon the simulation model established in Task 2 and insights gained from Task 3, this task will initially introduce simplifying assumptions to streamline the original, complex life-cycle cost simulation process. The aim is to identify simplifications that reduce the intricacies of the problem. Task 4 will focus on crafting an analytical model to estimate the life-cycle cost associated with system openness. To validate the proposed analytical model, simulation experiments will be designed and conducted using the virtual environment developed in Task 2 and fine-tuned in Task 3.

Chapter 2 Life-Cycle Cost Simulation

To build a simulation-based optimization (see footnote 1 in Chapter 1), a model that can convert the inputs including system architecture is first developed, refresh strategy and other system parameters into a life-cycle costs. The model is then implemented in a stochastic discrete-event simulation that can determine the difference between two different scenarios' life-cycle cost by generating the events and accumulating the associated discounted cost.

In this chapter, Section 2.1 provides an overview of the simulation process. Section 2.2 describes the inputs for the simulation. Section 2.3 discusses the cost model and how the contributing cost factors are evaluated. Section 2.4 proposes a relative life-cycle cost approach to represent the result of the simulation.

2.1 Simulation Overview

A discrete-event simulation has been developed as the basis for the modeling in this dissertation. Discrete-event simulation is commonly used to model life-cycle costs that are accumulated over time when time spans are long, and the cost of money is non-zero. In general, discrete-event simulator models a sequence of events along a timeline. At each event, various properties of the system can be calculated and aggregated. The simulation also incorporates probability distributions to represent the uncertainty in the simulation inputs. Consequently, the simulation creates the timeline (and accumulates relevant parameters) with many trials (i.e., through many possible time histories), aiming to build a statistical model predicting the life history of the system.

In the simulation developed for this dissertation (Figure 2.1) the events of interest are maintenance events, production events (delivery of new systems), retirement events (retirement of fielded systems), logistics events (spare parts management, lifetime buys for obsolescence mitigation) and design redesigns/refreshes. The accumulated property of interest is cost. The

model generates lists of events with specified dates and types of events. The event dates are determined by the simulation inputs, including sampling time-to-failure (TTF) distributions, forecasted obsolescence data distributions, and a predetermined refresh schedule. The event lists are then provided to a cost model to accumulate the discounted life-cycle cost.

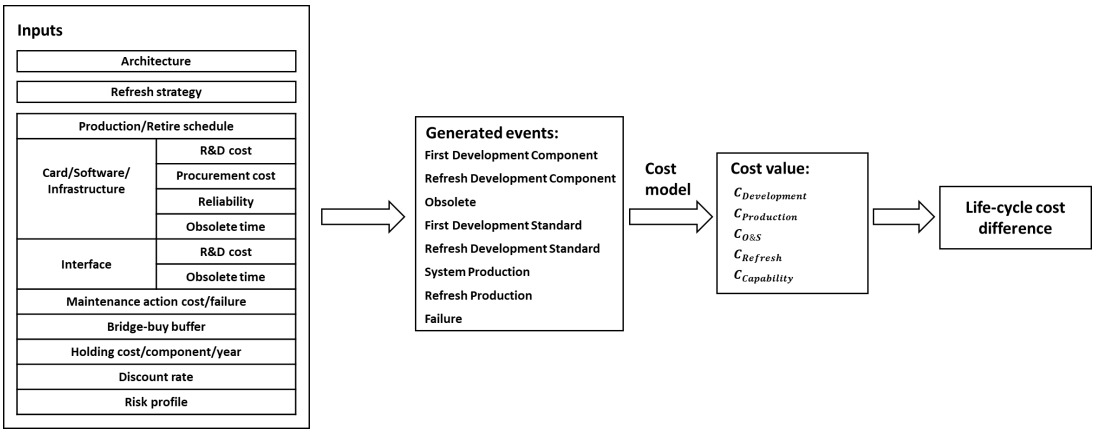


Figure 2.1 Simulation overview.

2.2 Cost Model

Figure 2.2 shows the structure of the life-cycle cost used in the model. As discussed in Section 1.6, the life-cycle cost refers to the "relative" life-cycle cost that is associated with the system openness. The life-cycle cost includes five cost categories: 1) Development/adoption cost, the non-recurring costs of system design, development and qualification. 2) Production cost, the recurring costs to manufacture and field the systems. 3) Refresh cost, the cost of implementing and qualifying technology refresh(es) during the system life cycle. 4) Operation and support (O&S) cost, the cost of sustainment incurred from system deployment to the end-of-support life. 5) Capability cost, the potential cost associated with the technology in the system being out-of-date relative to state-of-practice.

These cost categories were selected since the corresponding cost values are influenced by the degree of system openness. Ultimately, the difference in the life-cycle cost between cases with varying degrees of openness is the only thing that matters; therefore, some costs can be

omitted (i.e., they subtract out of the analysis). For example, the cost of fuel is one particular cost that may not be affected by openness, so it is not considered in our cost model. In this section, the details of each of the categories and how they are evaluated are discussed.

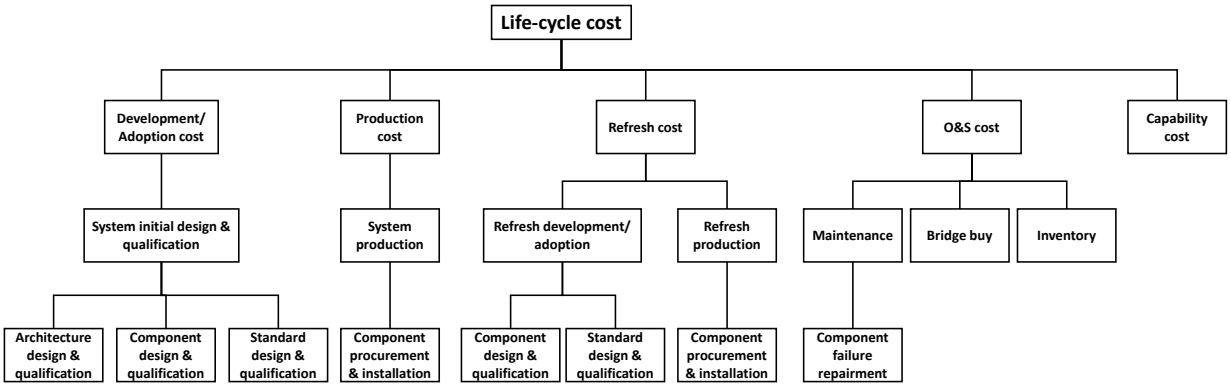


Figure 2.2 Cost model structure.

2.2.1 Development Costs ($C_{Development}$)

Development cost is the cost associated with designing a new system that satisfies a set of requirements and includes the majority of the costs incurred before the final design is selected, including the cost of designing the system architecture and components, adopting appropriate COTS parts and open standards, as well as the costs of partial or alternative designs considered, but not implemented. Prototyping and any design overhead costs are also included. Development cost should also include the Non-Recurring Engineering (NRE) costs, which may consist with testing and qualification costs to assure that the interfaces, components, subsystems, and complete system meet the required design parameters for performance, reliability, security, etc. These costs may be significantly lower for enterprises that maintain a library of previously used and qualified hardware and software components.

In our cost model, the development cost $C_{Development}$ is only incurred at the beginning of the timeline when the first version of the architecture was developed. It can be divided into three sub-categories corresponding to design/adoption and qualification of the three aspects of the system: architecture, component and interface.

The design and qualification cost of the architecture is related to the complexity and the openness of the architecture. For example, a more complicated architecture with more components and interaction within the architecture requires higher cost for design and qualification.

The design and qualification cost of components is the cost to design or select the components for the system. This cost depends on the type of components used in the architecture. Typically, using COTS components results in a smaller design cost than creating a proprietary component since the former can be acquired directly from the market. Each component undergoes qualification testing to ensure it meets performance and reliability criteria. In our model, each component is assigned design and qualification cost as inputs. The total adoption/development cost associated with the components is calculated by summing the design/qualification cost of each type of component that are used in the architecture.

The design and qualification cost of interfaces is the cost to select the open standards. Standards are adopted based on maturity, market acceptance, and potential availability for future system upgrades. Similar to the component design/qualification cost, the assumed inputs for this cost include the design and qualification expenses of each selected interface in the architecture.

The actual development process from architecture design, component and interface selection to the qualification may take several years to complete. The total development costs are charged at the starting point of the life cycle (at time zero).

2.2.2 Production Costs ($C_{Production}$)

Production cost includes all costs to manufacture and field the systems, including component procurement, assembly/manufacturing, stress screening of hardware components, and any recurring testing costs.

The production events are generated based on the system's production schedule. In our model, the production (recurring) cost of an instance system is defined as the sum of the costs of procurement, assembly, testing of all the system components and the installation of the final system in the field. The procurement cost of a component depends on the type of component.

2.2.3 Operation and Support Costs ($C_{O\&S}$)

Operation and support costs are the costs of sustainment incurred from system deployment to the end-of-support life, such system operating, modifying, maintaining, supplying, training, and inventory supporting. The events associated with sustainment include, but are not limited to: system failure repair, periodical maintenance, sparing management and obsolescence mitigation. In our model, the O&S costs is made up of maintenance, and spare parts management.

Maintenance events occur when a system failure occurs. Maintenance includes the labor to perform maintenance and the cost of spare components (if relevant). The model assumes that the downtime associated with a failure is negligible and the availability impacts are not included. The model also assumes that replacement components (spares) are good-as-new. If the component is still available in the market, it will be procured as needed. Therefore, the cost of the maintenance event before the component obsolescence is the component procurement cost plus the installation cost.

When the obsolescence of a component occurs, a bridge or lifetime buy is made to procure a sufficient number of components to support the system until the next system refresh or the end of system support, whichever happens the soonest – these components must cover future production and maintenance needs. The cost incurred at the obsolescence event is the procurement cost of the bridge buy of components.

Since bridge or lifetime buy components are purchased in advance and stored in inventory,

each component incurs an additional inventory cost that depends on the duration of its storage. Inventory (holding) costs are charged when the parts are taken from the inventory and used for maintenance. The holding cost of a component is the unit holding cost (per part per unit time) multiplied by the total time of the part stored in the inventory. The shelf life of all components is assumed to be long enough that parts bought in a bridge buy can be used for the rest of the system support life.

2.2.4 Refresh Costs ($C_{Refresh}$)

Through the entire life cycle, system refresh may be desirable (or necessary) due to the assurance of system supportable or the capability improvement of the system. System refresh replaces the obsolete or technologically out-of-date components with procurable and possibly more technologically advanced components. Refresh costs include the cost to develop or adopt the desirable components, the cost to deliver the refresh to the individual system, and the cost to re-qualify the system as needed.

In the cost model, refresh costs consist of two cost sub-categories: refresh development/ adoption and refresh production.

A schedule of periodical refresh development events is an input to the model. A refresh development results in a new baseline of architecture, with a new version of components or interfaces in the architecture. The system follows the current architecture baseline for production and refresh delivery until the next refresh baseline is developed.

At a refresh development event, every component and interface in the architecture is examined. If a component is obsolete or is required to be upgraded periodically, it will be refreshed. A refresh-required component might affect other components or interfaces to be refreshed as well due to ripple effect in the architecture.

In our model, a refreshed component is replaced with another component with the same

function, same support life (drawn from the identical time-to-obsolete distribution), and good-as-new reliability. If the refreshed component connects to its surrounding components with open standards, a "plug-and-play" refresh may be achieved, i.e., the component could be switched without affecting the surrounding components and standards and replace it with a new component that conforms to the open standard. On the other hand, if the connection between components is a closed interface or the open standard is obsolete, the new component may require that the surrounding components also have to be refreshed which we refer to as the ripple effect.

The structure of refresh development/adoption cost is similar to $C_{Development}$ described in Section 2.2.1 except that the design and qualification cost of the architecture is excluded from the refresh cost. Since it is assumed that the architecture itself does not change throughout the life cycle, there is no cost associated with the architecture design. Therefore, the refresh development/adoption cost is the sum of the design and qualification cost of the components and the standards that are required to be refreshed.

For refresh production, every fielded system adopts the same refresh production interval after its initial production. Therefore, the actual delivery date to each system of the refresh production might be different. For example, for a 4-year refresh schedule, a system fielded in year 0 would receive its refresh in years 4, 8, 12, etc. Another system fielded in year 3 would receive its refresh in years 7, 11, 15, etc. The version of components received for refreshes depends on the refresh development baseline available in the year of the refresh.

The refresh production cost has the same structure as production cost, including the cost of purchasing and assembly of the components to the system. The refresh production cost in each system refresh production is the total procurement cost and installation of the components that required to be delivered to the legacy system.

2.2.5 Cost of System Technological Capability ($C_{capability}$)

In general, capability is defined as the "the ability to achieve a desired effect under specified standards and conditions through combinations of means and ways to perform a set of tasks" (DoDAF, nd). For the purposes of this paper, the system's technological capability is defined as its ability to accomplish the "mission" it was designed for. For example, the absolute capability of a sonar system is its effectiveness in detecting objects in the surrounding area, while its relative capability is its effectiveness detecting adversaries early enough to take appropriate action. In the case of a sonar system, the system's technological capability is determined by performance parameters that include detection range, response time, detection accuracy, etc.

The cost of system technological capability is not just the cost to implement the capability, which is already included in the cost model in Section 2.2.1 to 2.2.4, but the potential costs that result from the capability (or lack of capability). More precisely, the cost is a result of the effectiveness of the system in performing the tasks required of it.

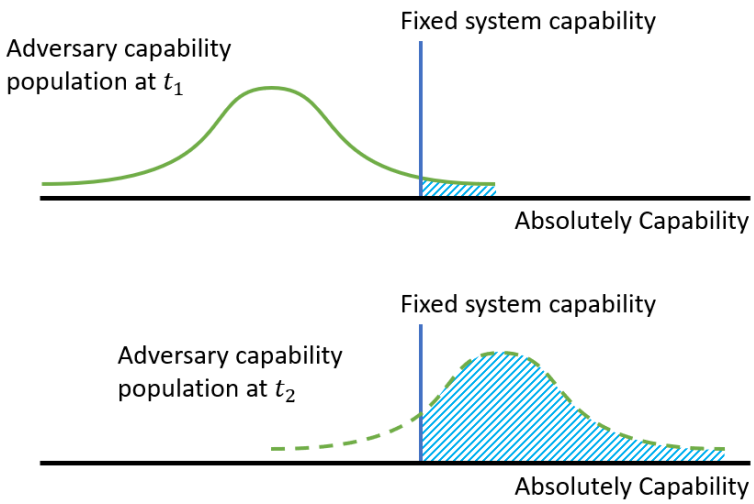


Figure 2.3 The adversary's absolute capability distribution shifts to the right over time relative to a fixed system capability.

For a system that is designed to operate in a competing environment such as defense, the cost of system technological capability is related to the system competitiveness among the

population of adversaries. Therefore, relative capability between competitors is a more appropriate metric to reflect the cost of system capability than absolute capability. In Figure 2.3, the distribution curves represent how an adversary population's capability evolves over time (from t_1 to t_2), and the vertical line indicates a fixed capability of a fielded system that does not receive any refreshes in the same timeframe. The population of adversary systems is represented by a distribution indicating the variance of the capability in adversary systems. The area under the distribution to the right of the fixed system capability represents the probability of the fixed system losing the capability competition to an adversary system. Figure 2.3 demonstrates that even if a system's absolute capability is fixed (i.e., doesn't change from t_1 to t_2), it may be losing its capability relative to the environment it is in. Decreases in relative capability represent a cost, which is either a decrease the effectiveness of the system in performing the mission it was designed to perform, or an increase in the risk of losing the system.[3] In this paper, the relative capability of a system instance is defined as the probability that the system capability is less than an adversary's system capability.

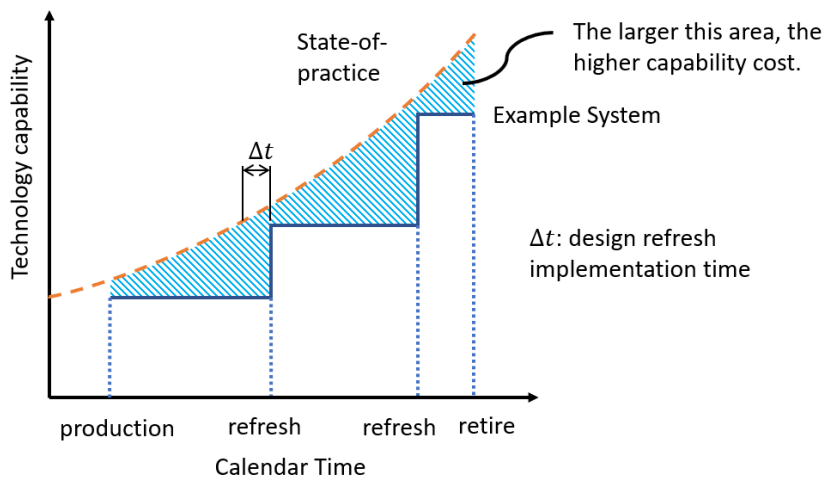


Figure 2.4 Relationship between system's technological capability and capability cost.

Figure 2.4 shows how the life-cycle capability cost can be evaluated based on the capability

[3] This means that even though the lowest life-cycle cost solution might be to field a system, make lifetime buys of parts as they become obsolete and never refresh the system, this may be an impractical solution for some systems because the relative capability of the system decreases over time.

gap between an example system and the state-of-practice of the technologies that comprise the system during the system's operational life. The shape of the state-of-practice curve in Figure 2.4 depends on the relevant technologies and their evolution over time. Where state-of-practice represents technology that has matured sufficiently to be practical for incorporation into relevant systems. Δt represents the technology lag time, which is the time it takes to implement a design refresh in the system (design, qualification, and fielding). The larger the Δt, the more likely a system will lose a capability competition. The area between the state-of-practice curve and the system capability curve is proportional to the total cost of capability during the system's operation life.[4]

As shown in Figure 2.4, each delivery of a technology refresh to the fielded system instance resets its capability to a higher level, closing the gap between the system and the state-of-practice capability. Frequent technology refreshes keep the system up-to-date during its life cycle, reducing the probability of losing the technology competition and decreasing the corresponding capability cost. Note, although the cost of relative capability is in the context of a defense application here, the concept is relevant to non-defense systems too. For example, a system whose competitiveness in the marketplace depends on constant technology refreshing – the risk articulated in Figure 2.4 could represent loss of market share.

Figure 2.4 provides a conceptual evaluation of capability cost. To obtain the quantitative cost, the relationship between capability and cost must be constructed. For a defense system, the cost of system capability may translate into risk, which evaluates the potential loss of systems and missions. Equation (2.1) provides the general formulation of the risk cost of a system instance in a given time window associated with capability.

[4] Figure 2.4 assumes that an adversary capability distribution shifts with state-of-practice. It is possible that the adversary is not bound by the same state-of-practice that is relevant for the system of interest, e.g., terrorists may use technology that has not matured sufficiently to be incorporated into traditional defense systems shifting the curve up from that shown in Figure 2.4.

$$R = IpC_q \tag{2.1}$$

In Equation (2.1), I represents the expected number of events (encountering an adversary system) in a time window, e.g., 2 times per year. The relative capability of a system instance p is defined as the probability that the system capability is less than an adversary system capability in this time window. C_q is the expected consequence (in cost) if a system loses the capability competition to the adversary.

Based on Equation (2.1), the calculation of total cost of system capability that is used in this study is given by,

$$cost\ of\ capability = \sum_{i=1}^{N}\sum_{j=1}^{T} Ip(\Delta t_{i,j})C_q \tag{2.2}$$

where N is the number of systems and T is the total number of support years for the fleet of systems.

The probability p of the ith system losing the capability competition in j^{th} year can be written as a function of technology lag time Δt. A discrete function $p(\Delta t)$ was first built based on the adversary technology evolution assessment process. The product $I\ p(\Delta t_{i,j})C_q$ models the expected cost in a one-year time window given the discrete technology lag time Δt.

Once the production profile and refresh interval are determined, the technology lag time $\Delta t_{i,j}$ of jth fiscal year of ith system instance can be evaluated. The annual monetary risk of each system can be calculated based on the annual risk cost function $Ip(\Delta t_{i,j})C_q$. The total cost of system capability is the summation of the risk cost of each system instance throughout their operation lives.

2.3 Relative Life-Cycle Cost Analysis Approach

It is often not practical to calculate the absolute value of all the life-cycle costs for a system.

The difference in the costs discussed in Sections 2.2 between the two system implementations is used to represent the cost of openness. This approach is referred to as a relative cost model (Sandborn, 2017). The advantage of a relative cost model is that all the costs that are a "wash" between the two architectures (i.e., the same) don't have to be modeled because they subtract out. The cost difference between two cases is significantly easier to determine than the absolute cost of the cases. This dissertation never produces absolute costs, only cost differences between two cases.

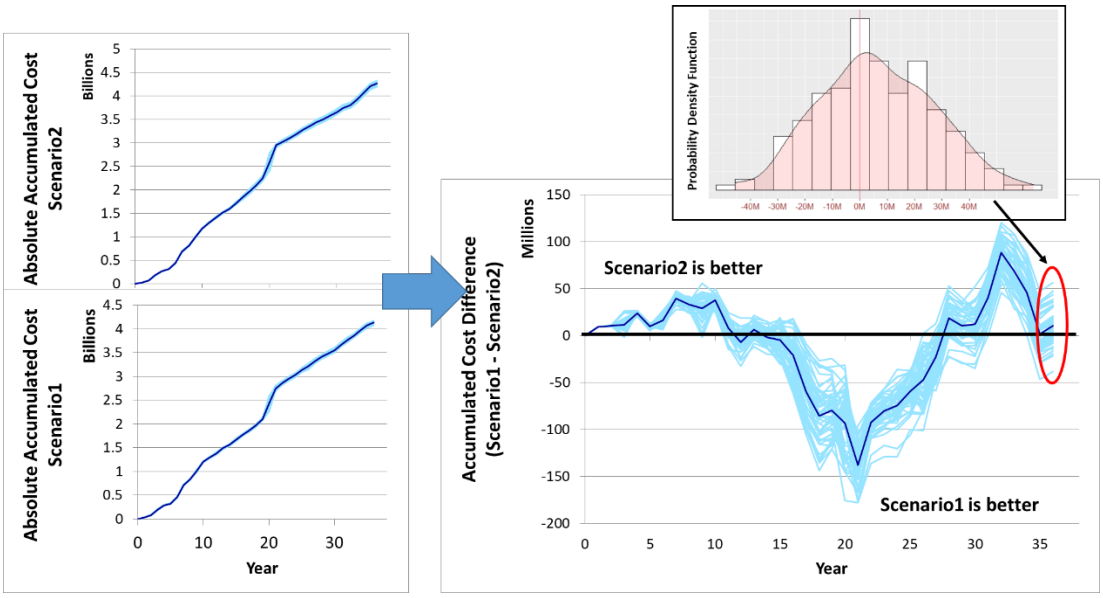


Figure 2.5 Sample solution construction of cumulative cost difference.

Figure 2.5 shows the process of obtaining relative cost, including an example result from the simulation model (the data used in this figure will be described in Section 3.1). On the left side of Figure 2.5 are the cumulative discounted life-cycle costs of two example scenarios calculated from the discrete-event simulation. In both cases many time-history solutions are shown (each is the result of a unique combination of samples from the input probability distributions, so each is one possible time history for the system). The darker solid blue lines are the mean of the time histories. The absolute accumulated costs for the two system configurations on the left side of Figure 2.5 are NOT particularly meaningful, because relevant

life-cycle costs that are not affected by system openness are omitted. The right side of Figure 2.5 shows the cost difference between the two scenario during the life cycle where the vertical axis is scenario 1 subtracted by scenario 2. The difference between the two cases is meaningful if the costs left out of the model are a “wash”, i.e., if they approximately subtract out. Figure 2.5 shows that at the end of 36 years, the mean value of accumulated cost different is greater than zero, indicating that the scenario 2 will have a lower life-cycle cost.

Chapter 3 A-RCI Case Study

In this chapter a case study using the detailed simulation developed in Chapter 2 of the Acoustic Rapid COTS Insertion (A-RCI) Sonar System is presented. The A-RCI program implemented a COTS-based open architecture into submarine sonar signal processing systems. The A-RCI eliminated traditional system architecture that used specialized proprietary components that were built to military specification, embracing the use of COTS and commercial standards, and allowed for the sonar signal processing system to be upgraded, without altering other sonar system components, (Guertin and Miller, 1998).

The transformation from a closed system to a COTS-based open system was completed in a four-phase implementation strategy (Guertin and Miller, 1998). In Phase I and Phase II, A-RCI developed a Multi-Purpose Processor (MPP) to process the data from both a towed array (TA) and a hull array (HA). Phase III added Spherical Array MPP (SA_{MPP}) and Switch MPP ($Switch_{MPP}$) to replace the legacy system spherical array processing functions. Phase IV integrated another high-frequency sail array MPP into A-RCI. By the end of Phase IV, a COTS-based open-architecture A-RCI system completely replaced the original legacy system.

In order to exercise the model described in Chapter 2, the life-cycle cost difference between two different A-RCI configurations with different degrees of openness implementation are examined in this Chapter.

The data describing the A-RCI in this section represents the author's interpretation of the A-RCI and the A-RCI program, and may not be representative of the actual system or program. The A-RCI is a defense system that spans many years, and as such, a complete data set describing the A-RCI is not publically available.

3.1 A-RCI Input Data

In this section, the data describing the A-RCI case is provided. Table 3.1 lists the input parameters used in the simulation for the A-RCI. Some inputs had to be assumed since there was no A-RCI specific information available, e.g., component procurement life. Some inputs were determined through calibration of the model against known A-RCI life-cycle costs, see Section 3.1.1.

Table 3.1 Input Parameters for Modeling the A-RCI

Input parameters		*Input value*	*Source*
Architecture		Figure 3.1	Guertin and Miller (1998)
Production schedule		Figure 3.3	Schuster (2007)
Retirement schedule		Figure 3.3	Schuster (2007)
Architecture R&D cost	Phase I	$48,350,000	From calibration, see Section 3.1.1
	Phase II	$39,015,000	
	Phase III	$55,745,000	
	Phase IV	$56,825,000	
Hardware: COTS cards	R&D cost per card type	$2,083,333	From calibration, see Section 3.1.1
	procurement cost per card	$7,331	
	installation cost per card	$733	
	reliability	Weibull (1.75,12)	
	procurement life	3 years	Assumed value
Hardware: proprietary cards	R&D cost per card type	$3,125,000	From calibration, see Section 3.1.1
	procurement cost per card	$14,545	
	installation cost per card	$1,454	
	reliability	Weibull (1.75,12)	
	procurement life	6 years	Assumed value
Hardware: infrastructure	R&D cost per infrastructure type	$1,000,000	From calibration, see Section 3.1.1
	procurement cost per infrastructure	$400,000	
	installation cost per infrastructure	$40,000	
	reliability	Weibull (1.75,30)	
	procurement life	20 years	Assumed value
Software	R&D cost	$12,500,000	From calibration, see Section 3.1.1
	procurement cost	$90,909	
	installation cost	$9,090	
	reliability	Weibull (1.75,12)	
	procurement life	3 years	Assumed value
Standards	R&D cost per standard type	$2,000,000	From calibration, see Section 3.1.1
	procurement life	10 years	Assumed value
Maintenance action cost/failure		$38,389	From calibration, see Section 3.1.1
Bridge buy buffer % of demand		50%	Assumed value
Holding cost/component/year		$1,000	Assumed value
WACC		5%/year	Assumed value

The architectural input data was based on Guertin and Miller (1998). Figure 3.1 shows the A-RCI architecture assumed in this analysis. A-RCI consists of four primary cabinets. Inside

each cabinet there are one or more VME drawers containing an array of cards. The software is connected to the cabinets with middleware. Other open standards are also represented by the linkages shown in Figure 3.1. The actual number of components (cards) is larger than those shown in Figure 3.1. The components that are common to all system architectures, i.e., they would be the same whether an open or closed system is assumed have been omitted. There is no need to model the common components since their impact on the life-cycle cost will subtract out of the relative cost model (see Section 2.3).

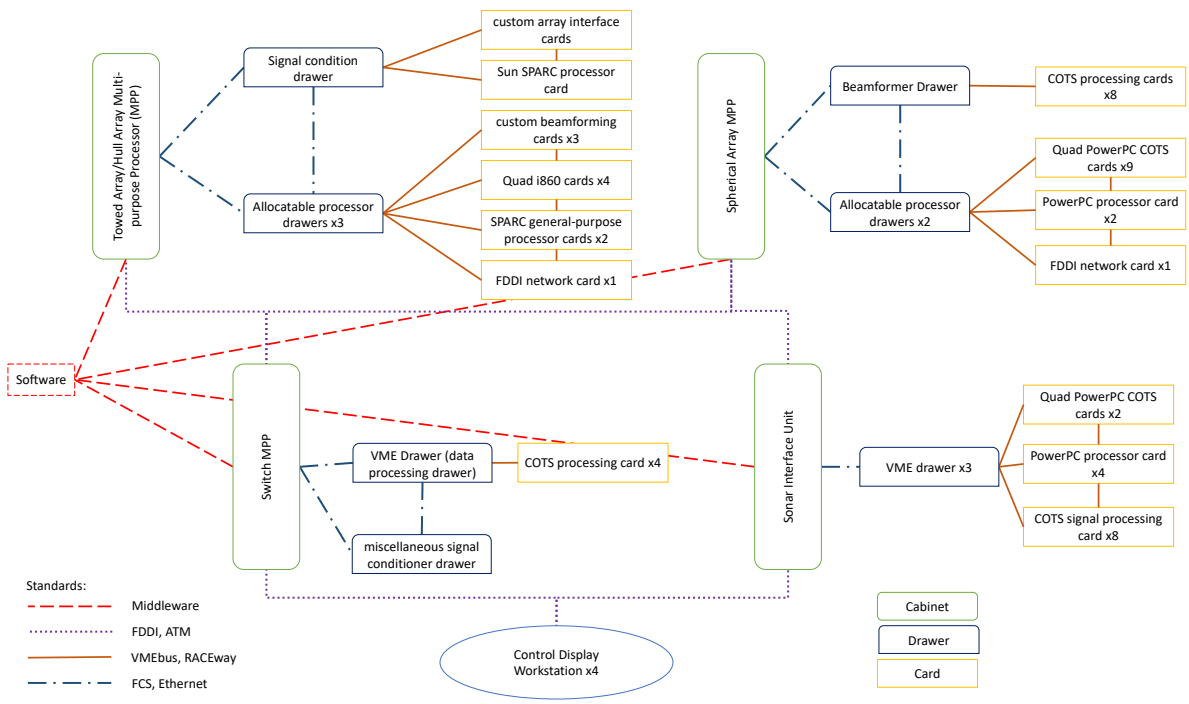
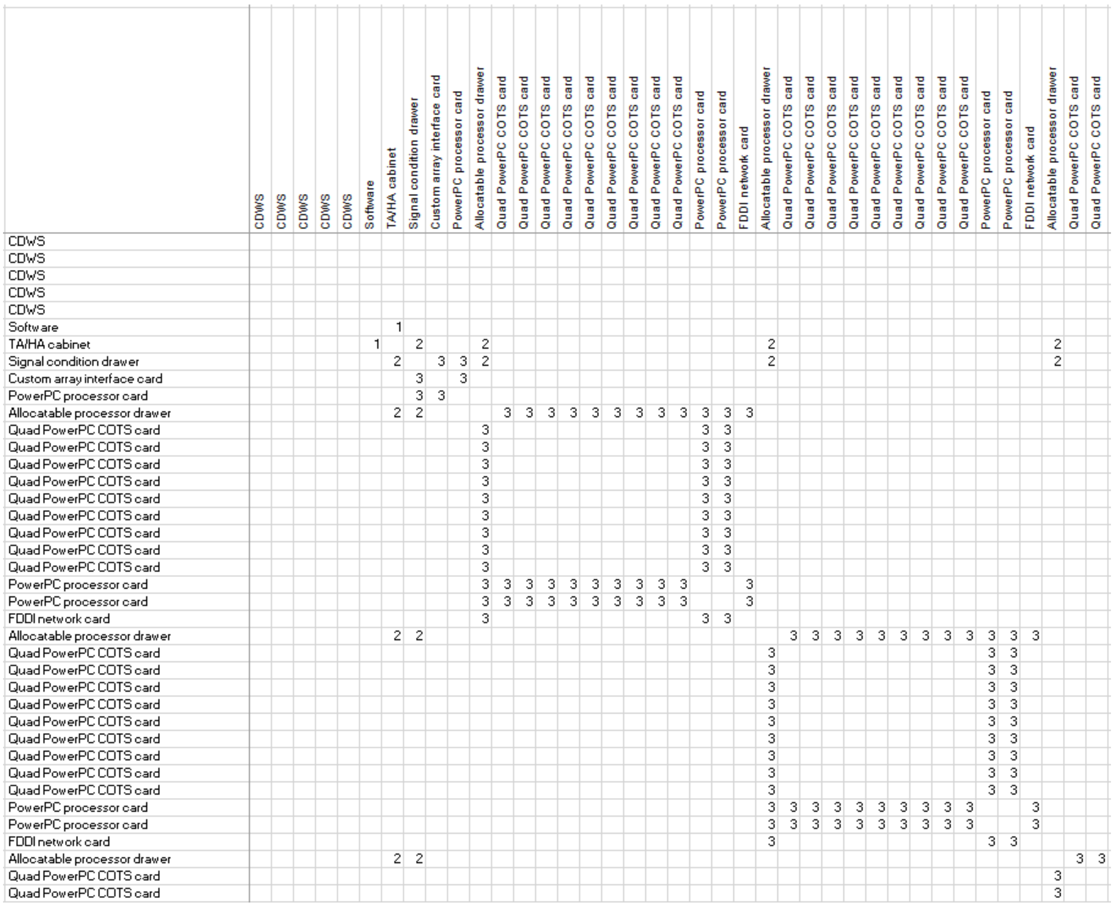


Figure 3.1 Architecture assumed in the A-RCI case study.

Due to the complexity of the A-RCI architectures, A design structure matrix (Eppinger and Browning, 2016; Paul, 1998) approach is selected to model the A-RCI architecture. The design structure matrix illustrates how the components inside the system are connected with other components via interfaces, capturing how the system's components interact with each other. Capturing the component interactions is necessary to cost the design refreshes, i.e., when a particular component is changed at a refresh, other required component changes can be determined by the design matrix; similarly, the interactions captured in the design matrix guide

re-qualification requirements for the system at refreshes.

Because of space constraints, Figure 3.2 only shows the portion of the design structure matrix of the A-RCI Phase III architecture that represents the architecture of TA/HA MPP. For an element a_{ij} in the ith row and j^{th} column, the value of the element indicates how component i interacts with component j. If a_{ij} is blank, there is no connection between components i and j. Different values of a_{ij} represent different interfaces between components i and j In Figure 3.2, the values 1, 2 and 3 represent different type of interfaces connecting components, which could have different times to obsolescence and development costs.



	CDWS	CDWS	CDWS	CDWS	CDWS	Software	TA/HA cabinet	Signal condition drawer	Custom array interface card	PowerPC processor card	Allocatable processor drawer	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	PowerPC processor card	PowerPC processor card	FDDI network card	Allocatable processor drawer	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	Quad PowerPC COTS card	PowerPC processor card	PowerPC processor card	FDDI network card	Allocatable processor drawer	Quad PowerPC COTS card	Quad PowerPC COTS card
CDWS																																							
CDWS																																							
CDWS																																							
CDWS																																							
CDWS																																							
Software							1																																
TA/HA cabinet						1		2			2													2													2		
Signal condition drawer							2		3	3	2													2													2		
Custom array interface card								3		3																													
PowerPC processor card								3	3																														
Allocatable processor drawer							2	2				3	3	3	3	3	3	3	3	3	3	3	3																
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
Quad PowerPC COTS card											3										3	3																	
PowerPC processor card											3	3	3	3	3	3	3	3	3	3			3																
PowerPC processor card											3	3	3	3	3	3	3	3	3	3			3																
FDDI network card											3										3	3																	
Allocatable processor drawer							2	2																	3	3	3	3	3	3	3	3	3	3	3	3			
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
Quad PowerPC COTS card																								3										3	3				
PowerPC processor card																								3	3	3	3	3	3	3	3	3	3			3			
PowerPC processor card																								3	3	3	3	3	3	3	3	3	3			3			
FDDI network card																								3										3	3				
Allocatable processor drawer							2	2																														3	3
Quad PowerPC COTS card																																					3		
Quad PowerPC COTS card																																					3		

Figure 3.2 Partial design matrix of the A-RCI Phase III architecture.

The installation, retirement profile and number of system instances of the A-RCI is shown in Figure 3.3, Schuster (2007).

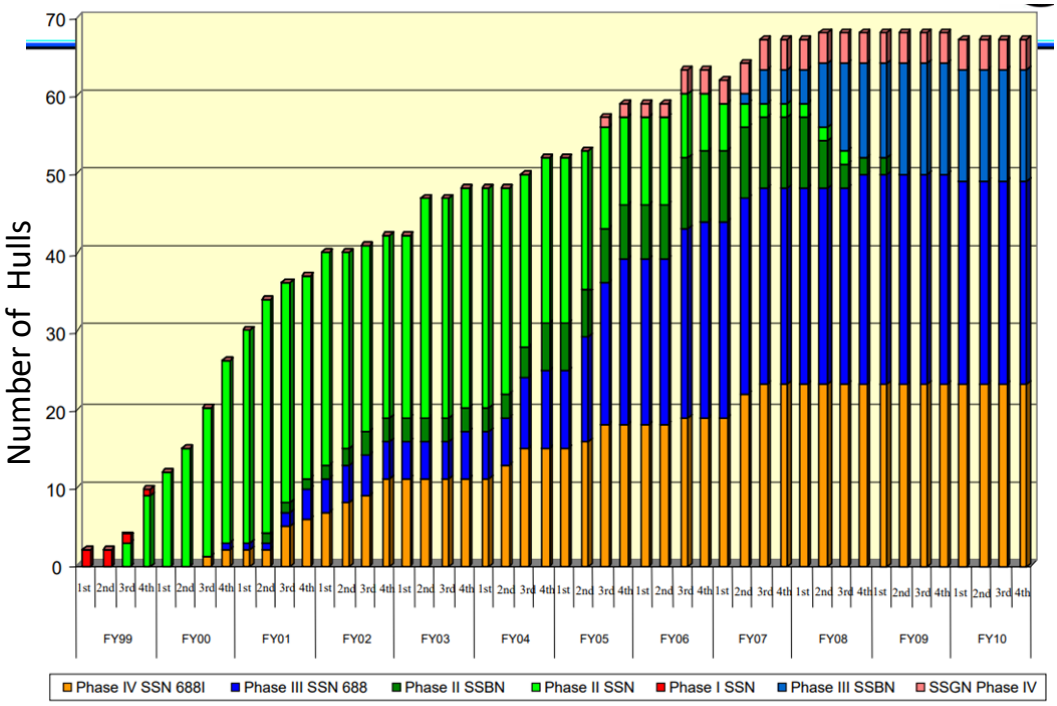


Figure 3.3 Installation profile for the A-RCI (Schuster, 2007)

3.1.1 Data calibration

The components and the interfaces that were used in the A-RCI were described in detail by Guertin and Miller (1998), however, the development and production cost, reliability and procurement life are not provided. These values are reverse-engineered based on the reported A-RCI life-cycle cost data.

The reported A-RCI life-cycle cost data was obtained from a 2006 presentation from ASSETT Corporation (ASSETT, 2006) that summarized the top-down and bottom-up cost estimations for the A-RCI system from 1996 through 2006. The ASSETT presentation provided the A-RCI annual cost data based on annual budget and contract data, which were mapped into three categories: Development cost, Production cost and O&S cost.

A simplified surrogate life-cycle cost model of intermediate variables was built for calibration. The intermediate variables included quantities such as average development cost of each phase and average production cost per system. After the intermediate variables were

calibrated to the known life-cycle cost totals, they were combined with other known information such as the number of components and architectures, to estimate the simulation model data. For example, the refresh development cost per year (intermediate variable) is determined based on the total development cost from ASSET and the refresh schedule used for the A-RCI. The refresh development cost per year (intermediate variable) divided by the number of components that were refreshed each year is the average refresh cost per component.

The detailed process of the calibration process is shown in Appendix A. The resulted parameters of the calibration process are included in Table 3.1.

3.2 A-RCI Modeling Results

In this section, two system configurations are compared based on their life-cycle cost. The first configuration matched the actual A-RCI architecture and refresh strategy. The second configuration was a less-open version of A-RCI where two of the modules adopted closed interfaces and proprietary components were used instead of COTS parts. Both configurations followed the same production/retirement schedule and used the same input data described in Table 3.1.

In the subsequent sections, an example simulation result shows the life-cycle cost difference as a function of end-of-support year. Selected sensitivity analysis is then performed to evaluate how three main key parameters, risk profile, end-of-support year and fleet size, affect the simulation results.

3.2.1 Example Result for Simulation Demonstration

Figure 3.4 shows the life-cycle cost difference as a function of end-of-support year. Each data point in the main plot of Figure 3.4 is a simulation result with the corresponding end-of-support year. The smaller plot on top shows the simulated annual accumulated cost difference of 36 end-of-support year. The right end results are mapped to the data points of the main plot where

end-of-support year is equal to 36.[5] In Figure 3.4, a positive result indicates that given the end-of-support life, the original A-RCI configuration is more beneficial, while a negative result value indicates that the less-open version of A-RCI is more beneficial. This result can be interpreted as: for the assumed risk profile, if the end-of-support year is between 35 to 39, the less-open version of A-RCI should be chosen instead of the original A-RCI.

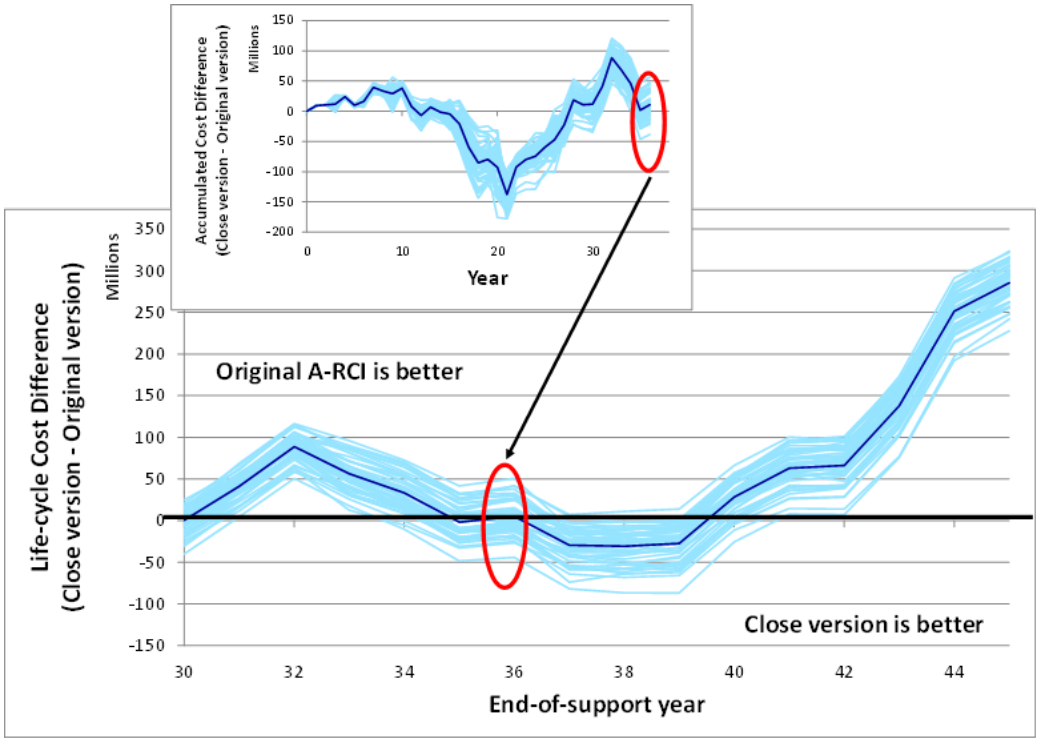


Figure 3.4 Life-cycle cost difference as a function of end-of-support year (expected consequence per capability competition loss is C_q = 1 million dollars, end-of-support = 36 years, 6-year refresh interval).

The results shown in Figure 3.4 represent one specific case to demonstrate the simulation result. In the sections that follow, a more detailed analysis comparing the open and closed versions of the A-RCI will be conducted.

3.2.2 The Effect of Refresh Strategy

In the previous section, the simulation is demonstrated by showing how it can be used to

[5] The accumulated cost difference in each year in the small plot does not represent the total life-cycle cost difference corresponding to an end-of-support year equal to that year. Varying the end-of-support year affects the cost of the events during the life cycle, resulting in different cost difference accumulations. For example, the number of components purchased in a lifetime buy is a function of the end-of-support year.

estimate the life-cycle cost difference between two system configurations. This section shows how the best refresh strategy should be determined based on its effect on the life-cycle cost.

Refresh strategy affects both refresh cost and capability cost. More frequent refreshing costs more but can keep fielded systems more up-to-date thereby reducing the cost of capability. For the same openness configuration, frequent refresh may be preferable when the system is used in a competitive environment.

The life-cycle cost of the original (open) A-RCI and the closed version are first compared assuming the same A-RCI refresh cycle, i.e., two-year refresh interval. The result in Figure 3.5 shows that the original A-RCI is always more beneficial than the close version (for a 2-year refresh interval).

The refresh strategy of the less open version of A-RCI is then varied but kept the original A-RCI configuration with the same original 2-year refresh interval. Three other refresh strategies were considered: no refresh, 6-year and 4-year refresh interval shown in Figure 3.5. In this case, no-refresh is the optimum refresh strategy for the close version of A-RCI since the curve is below the other cases and below 0 in Figure 3.5. The closed version of A-RCI is more beneficial than the original as the no-refresh curve is negative in the end-of-support year range from 30 to 45.

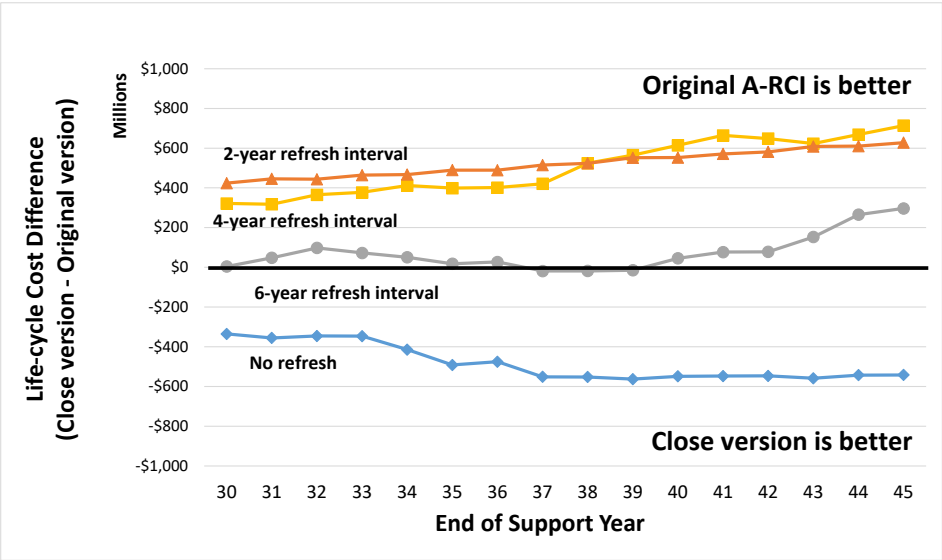


Figure 3.5 Life-cycle cost difference given different refresh strategies (C_q = $1M consequence cost assumed).

Figure 3.5 indicates that more frequent refresh increases the life-cycle cost difference between the closed version and the original version. As stated previously, refresh cost and capability cost are two cost factors that are directly influenced by the frequency of refresh. Frequent refresh results in high refresh cost but low capability cost. Figure 3.5 shows that for a $1M consequence cost, the increase in refresh cost is more significant than the decrease in capability cost.

3.2.3 Sensitivity Analysis

In addition to the length of life cycle (end of support year) and the refresh strategy, in this section, other external factors are analyzed and realized how they could affect the cost of openness. The risk profile and the fleet size are selected as two key factors in our case study and a corresponding sensitivity analysis was performed.

In the context of this case study, risk profile is characterized as the consequence of losing the capability competition to adversary systems and the frequency of encountering an adversary system. For the A-RCI, the risk profile likely varies based on adversarial conditions. If the risk profile is more severe (either a higher probability or a higher consequence), given the same

system capability, the capability cost is expected to be higher.

For the sensitivity analysis, the risk profile was varied by changing the expected consequence of losing the capability competition when encountering an adversary system. The expected consequence ranged from C_q = \$1 million to \$20 million per capability competition loss.

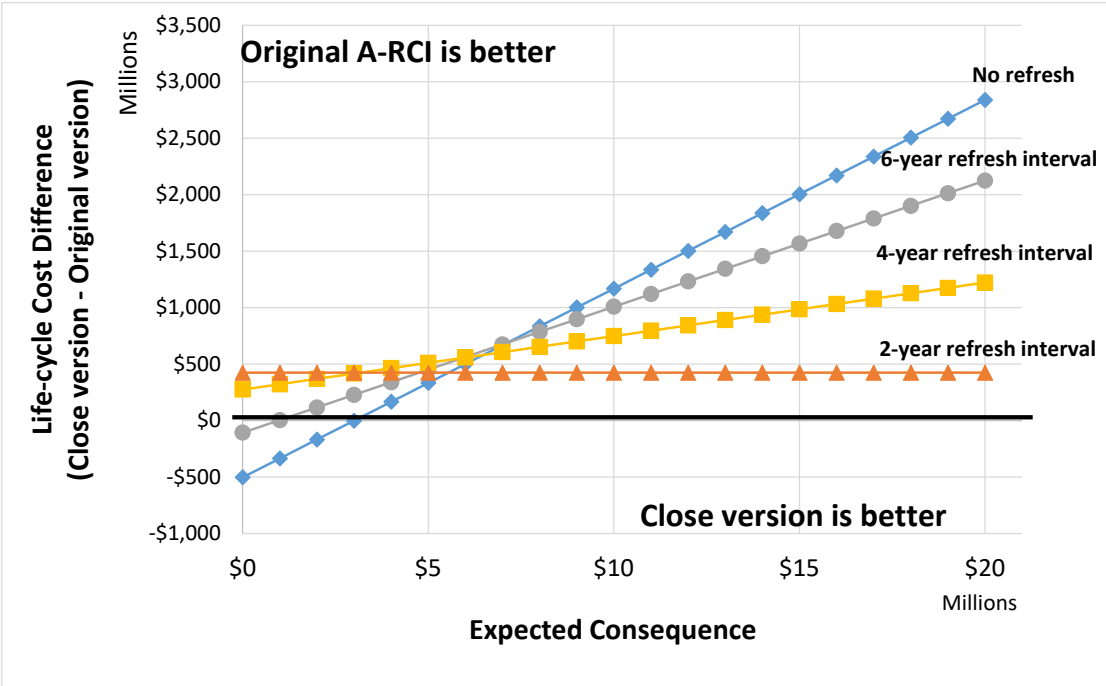


Figure 3.6 Cost difference comparison of 30-year support life given different risk profiles and refresh strategies.

Figure 3.6 shows the cost difference between the two configurations (the original A-RCI with its 2-year refresh and the closed architecture A-RCI with various refresh intervals) all of which have a 30 end-of-support year. Each data point is the mean value of one simulation cost difference result. The four curves shown represent different refresh strategies that were implemented to the second (closed) configuration. In Figure 3.6, the data points that have a positive cost difference are the results for which the original A-RCI configuration is more beneficial, and the negative cost difference points indicate that the less open version of A-RCI is preferred.

The curve of the 2-year refresh interval is a flat line in Figure 3.6. Since the two configurations both follow the same A-RCI refresh cycle, the technology performance would be the same throughout the life cycle in both configurations. Therefore, the life-cycle cost difference does not change over the expected consequence, i.e., there would be no sensitivity to the capability cost.

Figure 3.6 indicates how the risk profile and the refresh strategy jointly affect the result. First, the three curves in Figure 3.6 except the 2-year refresh interval are all increasing functions, indicating that no matter what refresh strategy the closed version A-RCI adopts, the open configuration becomes more beneficial as the expected consequence increases. However, since the original A-RCI has the most frequent refreshes, the increase in capability cost of the original A-RCI is less than the increasing in capability cost in the close version A-RCI, resulting in positive slopes of the three curves. Second, the slopes of the three curves decrease as the refresh becomes more frequent. The life-cycle cost of the closed A-RCI with no refresh is more sensitive to the risk. Among the three refresh strategies, when the expected consequence is less than $6 million, the no refresh strategy is the optimum strategy. 2-year interval refresh becomes preferable when the expected consequence is more than $6 million. Lastly, once the expected consequence is over $3 million, it is always more beneficial to adopt an open system approach rather closed approach. In conclusion, as the expected consequence increases, capability cost becomes dominant. Therefore, in order to reduce the capability cost and have a lower cost per refresh, an open configuration with frequent refresh is favorable.

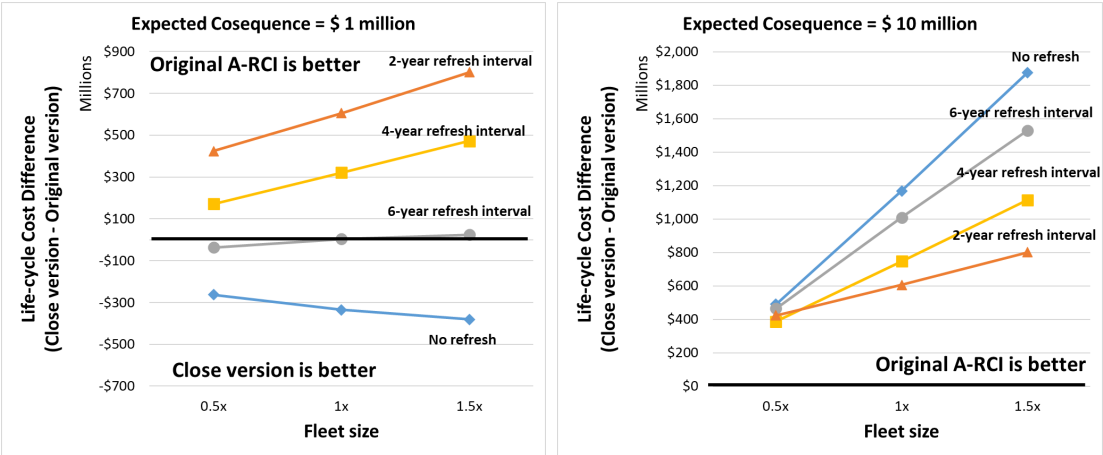


Figure 3.7 Cost difference comparison of 30-year support life given different fleet sizes and refresh strategies.

The number of fielded systems is another factor that potentially impacts the results. Recurring costs, e.g., production cost and refresh delivery cost, and capability cost are directly influenced by the number of fielded systems. Figure 3.7 shows two sensitivity analyses of the number of fielded systems under low and high-risk profiles separately. In Figure 3.7, all the systems were supported for 30 years and three scenarios were considered: 0.5, 1 and 1.5 times the original A-RCI fleet sizes. As the fleet size grows, the influence of the fleet-size-dependent cost factors have more impact on life-cycle cost. These factors include system production cost, system refresh delivery cost and capability cost and the most dominant cost factor determines whether an open system is favor or not. On the left side of Figure 3.7, capability cost is slightly more dominant in 2-year, 4-year and 6-year refresh interval. The original A-RCI configuration with more frequent refresh and less capability cost therefore becomes more favorable as the fleet size increases. For no refresh strategy, refresh delivery cost is more dominant, so the closed version with no refresh and less refresh delivery cost becomes more favorable over the fleet size. On the right side of Figure 3.7, the expected consequence is increased to $10 million, leading to a significant capability cost. Thus, the original A-RCI is always better and the effect of capability is magnified as more systems are fielded.

3.3 Conclusion of the A-RCI Case Study

In this chapter, the A-RCI program was selected as a case study to perform the life-cycle cost difference analysis. Since most of the cost data of A-RCI which was required as the inputs for the simulation was unavailable to the public, a calibration process was first conducted to determine the input values for the simulation. In the life-cycle cost difference analysis, the original A-RCI architecture with the original refresh strategy was set as the baseline configuration. A less open A-RCI architecture was created. The refresh strategy of the less open version of A-RCI was altered to see how the refresh strategy influences the life-cycle cost difference. In addition, the contributing factors of the expected consequence and the fleet size were also selected to perform a sensitivity analysis.

Based on the result of the simulation, a higher system openness does not always guarantee cost effectiveness. The life-cycle cost of a system is a result of the combination of system architecture (openness), refresh strategy, environmental factors (consequence of losing capability), fleet size, etc. For example, in a competitive environment where the consequence of losing capability is high, it might be better to have an open system with more frequent refresh. On the contrary, a less open system with no refresh might be more suitable during peace time. In conclusion, in addition to system openness, the factors mentioned in this chapter should all be considered when making a decision to reduce life-cycle cost.

Chapter 4 Generalized Open Systems Life-Cycle Cost Model

In the preceding chapters, a discrete-event simulation approach for modeling life-cycle cost differences related to system openness was introduced. While this simulation offers a detailed analysis of cost avoidance, it faces practical challenges. Firstly, it demands an extensive amount of detailed input data, from the individual component/interface procurement, development, qualification costs and support lives to the architecture development and qualification costs, often unavailable during conceptual design. Secondly, the simulation's execution time can be prolonged, especially for complex systems like A-RCI, taking hours to complete a simulation. Given that conceptual design prioritizes identifying a system openness direction rather than detailed cost analysis, the simulation approach may prove too cumbersome for practical use.

To address these challenges, there is a need for a more generalized model that establishes a relationship between cost and system openness. This model, based on simplified assumptions, would serve to provide a preliminary understanding of the system's desired level of openness. Moreover, it could yield simple guidelines or "rules of thumb" for developing an open system, aligning with the broader goals of conceptual design.

An example of a generalized model is the COTS-LIMO model proposed by Abts (Abts, 2000). This model hypothesizes how the maintenance cost of a COTS-based software system is affected by time and the number of COTS components in a system. The COTS-LIMO model is a conclusion from the COCOTS model that is a life-cycle cost model based on data interviews from the software industry.

The COTS-LIMO model, postulates how the number of COTS components in a software system could be determined based on the maintenance cost criterion and sustainment life. This model assumes that the maintenance cost for a COTS-based system increases over time. Since

the COTS components may evolve in different directions in response to the market, the functionalities and linking interfaces of the COTS components might change over time and become different from the original COTS components that were selected for the system. Therefore, a system with more COTS components requires more effort to manage the changes of COTS components over time in order to maintain system operation. The number of COTS in the system determines the time rate of change of the cost. There is a threshold for the cost of maintenance, and if the cost of maintenance exceeds the threshold, the system becomes too expensive to sustain and should be retired. The COTS-LIMO model infers the following rule for COTS-based software system design:

Maximize the amount of functionality in your system provided by COTS components but using as few COTS components as possible.

This rule, called the CBS Functional Density Rule of Thumb (Abts, 2000), is generally accepted, but has not been quantitatively tested. In a doctoral dissertation by Dunn (Dunn, 2011), the interviews of practitioners from IBM (International Business Machines) showed that the practitioners believed that system costs could be reduced by building more of a system from COTS components since the development effort would be 'time-shifted' to the vendor. However, statistical analysis from the same study that collected the raw data from IBM, indicated that there was no significant correlation between the percentage of COTS components[6] and development and maintenance effort.

Although the validity of the CBS Functional Density Rule of Thumb has not been established, it provides a possible direction for the development of a generalized model for life-cycle cost. This dissertation aims to construct a surrogate model that is based on the previous sophisticated simulation model to estimate the life-cycle cost associated with the system

[6] To be more specific, it is the percentage of system functionality provided by COTS components.

openness. The aim of this generalized model, called Generalized Life-Cycle Cost (GLCC) model, is to employ a reduced set of representative inputs for a more practical application. Additionally, by utilizing the analytical capabilities of the generalized model, the computation time and the number of detailed inputs required for the original DES simulation can be substantially decreased. The objective of the GLCC model is to enable the prediction of the optimal level of openness for an architecture.

This chapter commences by introducing two novel approaches: the GLCC model and the GLCC predictor, as outlined in Section 4.1. This section emphasizes their relationship between the simulation model, detailing the inputs and anticipated outcomes. Chapter 4 places a primary emphasis on the GLCC model. Sections 4.2 through 4.5 offer a comprehensive explanation of the GLCC model, encompassing its assumptions, inputs, and intricate model details. The subsequent sections delve into the validation of the GLCC model through a comparative analysis with the simulation, addressing both the simplification process and validation outcomes. Finally, Section 4.8 unfolds a practical application of the GLCC model in the A-RCI case, offering a real-world example to illustrate its effectiveness.

4.1 Introduction

Figure 4.1 depicts the relationship between the simulation, GLCC model, and GLCC predictor. As discussed in Chapter 2, the simulator was introduced to simulate the life-cycle cost (difference) using detailed inputs. However, the simulation process is time-consuming, requiring substantial time to obtain results for a single openness configuration. Even though the simulation can yield intricate results with a more detailed setup, the time constraint makes it impractical to perform multiple scenarios or identify the optimal openness configuration efficiently, i.e., it is not practical for use in evaluating conceptual design tradeoffs.

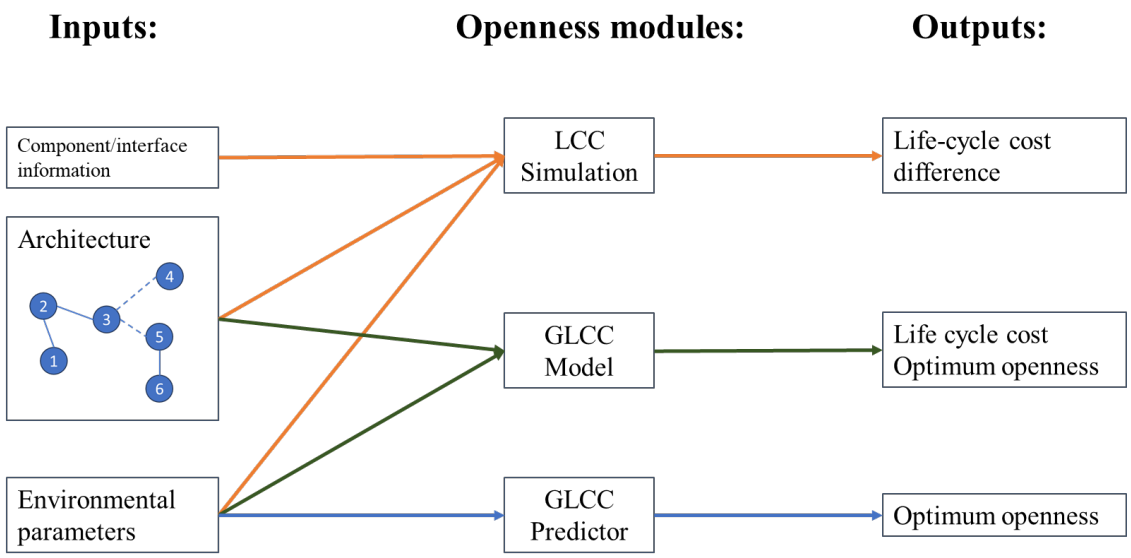


Figure 4.1 The three life-cycle cost approaches, simulator, GLCC model and GLCC predictor.

The original discrete-event simulation requires three types of inputs, the component/interface information, the system architecture and the environmental parameters[7]. They are shown on the left side of Figure 4.1 (not all of which are necessary for the GLCC model and the GLCC predictor). Firstly, the component/interface information specifies the openness of individual components and interfaces within the architecture network. It indicates whether each component or interface is open or closed. The system architecture represents a network that outlines the topological relationship between components (nodes) and interfaces (edges). The remaining inputs are categorized as environmental parameters, encompassing details such as the development/procurement cost, support life of components/interfaces, end-of-support year, discount rate, refresh review interval, and others.

The inputs of the GLCC model are the architecture network and the environmental parameters. The GLCC model combines network analysis and analytical solutions, incorporating certain simplifying assumptions into the model. The GLCC model is capable of

[7] The inputs discussed here remain consistent with those outlined in Chapter 1. Component/interface information and system architecture correspond to the system architecture g, while the refresh strategy s and the parameter set p are combined as the environmental parameters in this context.

exploring all possible openness configurations for the architecture to find the optimum openness for the system architecture.

The GLCC predictor introduces additional simplifications. While the GLCC model is utilized to ascertain the optimal level of openness through relative life-cycle cost analysis, the GLCC predictor primarily focuses on predicting the trends in the usage of open/closed components/interfaces based on predefined environmental parameters. Notably, the GLCC predictor operates with the fewest input requirements among the three approaches. While it may not provide the precise determination of the optimal level of openness, as the GLCC model does, it can predict the level of openness with the minimum cost among the four scenarios. The following options encompass various combinations that can be considered: [open components, open interfaces], [open components, closed interfaces], [closed components, open interfaces], and [closed components, closed interfaces]. This predictive capability serves as a valuable practical guideline. Furthermore, the GLCC predictor, relying exclusively on analytical calculations, is the least computationally intensive method for generating results.

4.2 Assumptions for Generalized Life-cycle Cost Model

Similar to the simulation model, only the relevant costs influenced by the system openness are included in the GLCC model. In other words, the "wash" costs between the open and closed system configurations are eliminated from the model. Therefore, the model does not produce accurate actual costs for either case, but the cost difference between the cases is meaningful, which allows the optimum openness for the system architecture to be found.

As discussed in the preceding section, the GLCC model deviates from the previous approach of generating discrete events used in simulations. Instead, it employs network analysis and analytical solutions to compute the life-cycle cost. Several assumptions play a crucial role in reducing the complexity of the life-cycle cost problem so that the integration of network

analysis and analytical solutions becomes feasible for the GLCC model.

1. Simplify the system production and retirement schedule. In the simulation, each system can be individually assigned a specific production year and retirement year, allowing for variations across systems. However, in the GLCC model, a simplifying assumption is made that all systems are produced in year 0 and remain in service without retirement throughout the support life.

2. Exclusion of capability cost. In the simulation, the capability cost represents the competitiveness of the systems and is determined based on the refresh frequency of those systems. To streamline the problem scope, the GLCC model adopts a simplified approach by assuming a constant refresh schedule. That is, when the GLCC model compares the life-cycle cost associated with different levels of openness, the refresh cycles are assumed to be the same. Therefore, the capability cost becomes identical between the cases considered and is therefore ignored in the model.

3. Simplified refresh policy options. In the simulation, the refresh of components can be driven by either the need to mitigate obsolescence or the desire to enhance capability. The former implies that components are refreshed only when they become obsolete, whereas the latter mandates component refreshes to fulfill specific capability requirements in order to attain a desired level of competitiveness. The GLCC model assumes that the refresh policy is based exclusively on obsolescence mitigation.

4. Universal refresh review interval. While the simulation had the capability to assign distinct refresh schedules to individual components or interfaces, the GLCC model uses a uniform refresh review interval that applies to all components and interfaces. This means that all components and interfaces undergo refresh evaluations at the same regular intervals. For example, with a predetermined T-year refresh review interval, the GLCC

model assesses whether components and interfaces need refreshing every T years.

5. Remove parameter uncertainty: Unlike the simulation model, which involves generating discrete events based on time-to-failure and time-to-obsolescence distributions, the GLCC model simplifies by eliminating the uncertainty associated with these parameters. Deterministic values are used instead. While this approach may sacrifice the representation of uncertainty in the life-cycle cost results, it facilitates easier analytical calculations of the life-cycle cost.

6. Simplification of refresh design and deployment. In the simulation model, it is possible to assign different refresh design cycles and refresh deployment cycles, that is, the time to develop a new system architecture baseline and the time to deploy the new architecture to the systems can be different. However, when transitioning to an analytical framework, keeping track of component numbers, versions, instances, and locations becomes challenging. To address this complexity, the GLCC model assumes that design refresh and deployment occur simultaneously. In other words, at the start of each refresh cycle, the development and installation of the refreshed system are considered complete instantaneously.

4.3 GLCC Model Inputs

The GLCC model utilizes two types of inputs: the topology architecture network and the environmental parameters. As discussed in the previous section, the system architecture can be represented as a network consisting of nodes and edges. The nodes correspond to components, and the edges represent interfaces. In certain cases, nodes can also represent modules, which are combinations of multiple components.

The environmental parameters are constant values associated with the component, interfaces, and systems. Each component has development cost (DC), procurement cost (PC), and support

life (O), while each interface has development cost (DC) and support life (O). Other environmental parameters include the number of systems (N_{sys}), end-of-support year (EOS), holding cost ratio (C_h), discount rate (r), failure rate (λ), bridge buy ratio (B), and refresh review interval (T).

The number of systems (N_{sys}) refers to the total number of systems to be supported and produced at the beginning of year 1 ($t = 0$). It is assumed that no systems are retired throughout the life cycle until the specified end-of-support year (EOS). The holding cost ratio (C_h) represents the inventory cost associated with holding one component. The holding cost of a component in a given time is assumed to be a fraction of the procurement cost of the component. In the GLCC model, the holding cost of a component per year is the holding cost ratio times the procurement cost of the component. The discount rate (r) is utilized to discount future costs back to the beginning of year 1 ($t = 0$). The failure rate (λ) indicates the expected number of component failures per year. When a component becomes obsolete, a bridge buy is performed to acquire an adequate quantity of replacement components to resolve all future component failures. The bridge buy ratio (B) denotes the ratio of the total quantity of bridge buy components to the estimated spare demands, where the estimation is based on projected failures occurring between the component's obsolescence and the subsequent refresh. The predetermined refresh review interval (T) is the duration between one refresh assessment and the next refresh assessment. Table 4.1 presents the complete set of inputs for the GLCC model.

Table 4.1 The Inputs of the GLCC Model.

Inputs	Description
EOS	End-of-support year
N_{sys}	The number of the supported systems
DC_i	The development cost of the ith component/interface
PC_i	The procurement cost of the ith component/interface
λ_i	The failure rate of the ith component/interface
r	Discount rate
T	The predetermined refresh review interval
O_i	The support life (time to obsolete) of the ith component/interface
B	The bridge buy ratio
C_h	The unit holding cost ratio per component per year
g	System architecture (in network)

4.4 GLCC Model Details

The GLCC model illustrates the relative cumulative life-cycle cost required to sustain a specific number of systems (N_{sys}) from the beginning of year 1 ($t = 0$). to the end-of-support year (EOS). This all-encompassing life-cycle cost covers the expenses associated with maintaining all components and interfaces of the systems throughout their end-of-support years. This holistic cost can be represented as the summation of the individual life-cycle costs of each component and interface, as demonstrated in Equation (4.1).

$$LCC = \left(LCC_{comp1} + LCC_{comp2} + LCC_{comp3} \dots LCC_{compn}\right) + \left(LCC_{inter1} + LCC_{inter2} + LCC_{inter3} \dots LCC_{interm}\right) \tag{4.1}$$

$$LCC = \sum_{i=1}^{n_c+n_I} LCC_i \tag{4.2}$$

Additionally, Equation (4.1) can be equivalently expressed as Equation (4.2), where n_c and n_I signify the total quantities of system components and interfaces, respectively. Within

Equation (4.2), the lowercase italic "i" denotes the specific component or interface (i^{th}) within the system architecture. This notation will be consistently applied throughout Chapter 4 for clarity and consistency.

Figure 4.2 displays the cash flow diagram illustrating the life-cycle cost of component/interface i. The life-cycle cost (LCC_i) of individual components and interfaces is a combination of non-recurring and recurring costs. The non-recurring cost of the ith component/interface, denoted as $Cost_{NRE,i}$, is incurred at the beginning of the component's or interface's life cycle, specifically at time 0. On the other hand, the recurring cost in the kth refresh cycle of the ith component/interface, represented by $Cost_{R,k,i}$, is accrued at the conclusion of the kth refresh cycle.

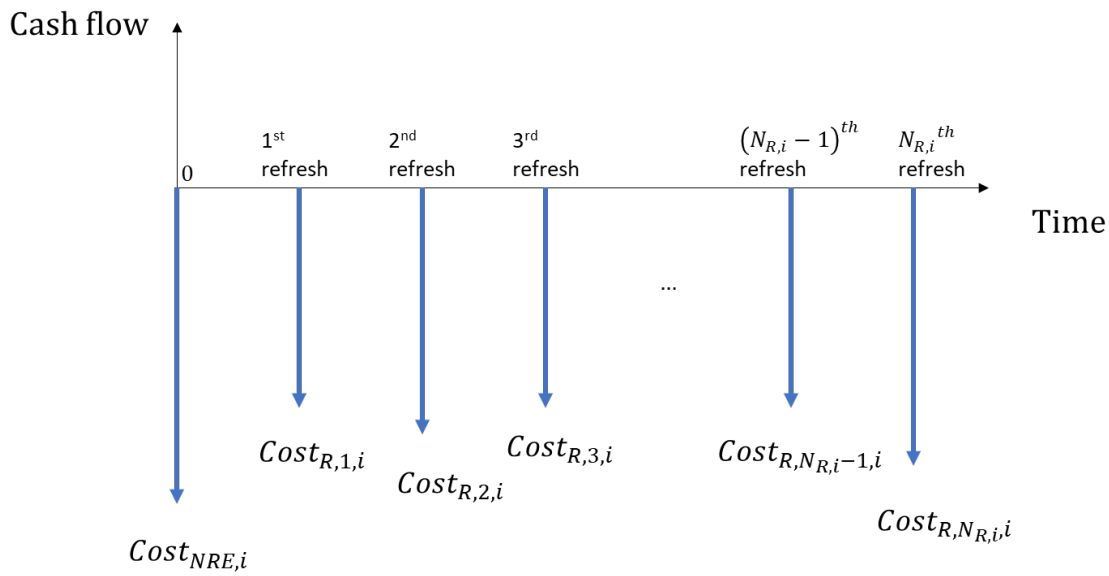


Figure 4.2 The life-cycle cost cash flow diagram of the ith component/interface.

Within Figure 4.2, $N_{R,i}$ signifies the total number of refreshes [8] experienced by component/interface i throughout its entire life cycle. This value can fluctuate from one component or interface to another. As previously mentioned, the GLCC model assumes a

[8] The actual total number of refreshes may not be the same as the total number of refresh reviews, since one component may not need to be refreshed after a refresh evaluation.

universal uniform refresh review interval for all the components and interfaces. This implies that the components and interfaces are evaluated for refresh requirements at consistent intervals. However, this does not necessarily imply that all the components and interfaces require an identical number of refreshes throughout their life cycle. (That is, $N_{R,i}$ may not equal to $N_{R,i+1}$) To illustrate, a component with a longer support life, i.e., longer time to obsolete, may necessitate fewer refreshes over its life cycle, as it may not require refreshes during every scheduled evaluation.

Furthermore, it is important to note that the time interval of each refresh cycle may not be the same throughout the life cycle of the component/interface. That is, the interval [0,1st refresh] may not be the same as the interval [1st refresh, 2nd refresh] in Figure 4.2. A component refresh may be triggered either by the obsolescence of the component or by the refresh of the connected component/interface, leading to instances where a component/interface may not require refreshing at every review. The determination of the refresh schedule for the components/interfaces is based on a proposed architecture network analysis and will be discussed in the subsequent section.

Equation (4.3) gives the present value of the cash flow diagram presented in Figure 4.2. Specifically, it reflects the combined sum of recurring costs and non-recurring costs. The recurring costs, observed at the end of each refresh cycle, are adjusted by a discount factor $f_{k,i}$ to map the costs to present value.

$$LCC_i = Cost_{NRE,i} + \sum_{k=1}^{N_{R,i}} f_{k,i} Cost_{R,k,i} \tag{4.3}$$

$$Cost_{NRE,i} = C_{Development,i} + C_{Production,i} \tag{4.4}$$

$$Cost_{R,k,i} = C_{Refresh,k,i} + C_{Maintenance,k,i} + C_{BridgeBuy,k,i} + C_{Hold,k,i} \tag{4.5}$$

Equations (4.4) and (4.5) provide a more detailed breakdown of non-recurring and recurring

costs. The non-recurring cost for the ith component/interface, designated as $Cost_{NRE,i}$, comprises two primary components: development ($C_{Development,i}$) and production ($C_{Production,i}$) costs. These costs occur at the initiation of the life cycle ($t = 0$) and therefore do not need to be discounted.

The recurring costs for the ith component/interface in the kth refresh cycle, represented as $Cost_{R,k,i}$, encompass four distinct recurring cost elements: refresh ($C_{Refresh,k,i}$), maintenance ($C_{Maintenance,k,i}$), bridge buy ($C_{BridgeBuy,k,i}$), and holding ($C_{Hold,k,i}$) costs. These subcategories are valued as future expenses, assessed at the conclusion of the corresponding refresh cycle.

To enhance clarity, consider a scenario where a component is first produced/procured at year 0 and undergoes its first three refreshes at years T_1, T_2, and T_3. The 1st refresh cycle spans from year 0 to year T_1, the 2nd refresh cycle from year T_1 to year T_2, and so forth. In the context of the 1st refresh cycle, the recurring cost includes the refresh cost at year T_1, in addition to the cumulative maintenance, bridge buy, and holding costs incurred within the time frame from year *0* to year T_1.

The subsequent sections will delve into the details of these cost element parameters and elucidate their computation based on the provided input parameters, further clarifying their role within the analytical cost function of the GLCC model.

4.4.1 Development Costs ($C_{Development,i}$)

The term "Development" encompasses the allocation of resources and efforts dedicated to activities like component/interface design, selection, qualification, testing, and even prototyping. To encapsulate all these costs, we use the development cost of component/interface i, denoted as DC_i.

It's important to recognize that system development goes beyond the realm of individual components or interfaces. For example, designing the system architecture itself requires a significant amount of effort. However, within the framework of the GLCC model, we assume

that the systems being compared in terms of their life-cycle costs already share the same system architecture, including the same network topology. As a result, the expenses associated with designing the entire architecture are considered identical for all systems and are omitted from the development cost model.

Within the GLCC model, the development cost of component/interface *i* is directly aligned with the input provided for the development cost of the ith component/interface, Equation (4.6)[9].

$$C_{Development,i} = DC_i \tag{4.6}$$

4.4.2 Production Costs ($C_{Production,i}$)

In the simulation, the production costs were charged based on the system production scheduled. For simplification, all the production cost of the GLCC is assumed to occur at the beginning of the system life cycle.

In reality, the production process should include the component's procurement, qualification and installation, and the testing of the system function. In the scope of this study, the procurement cost of the components differs, but the rest of the efforts are assumed to have similar cost. Therefore, the production cost of component/interface *i* only considers the procurement cost, which is the product of procurement cost (PC_i) and the total number of systems (N_{sys}), as shown in Equation (4.7) In addition, since the interfaces do not have physical shape, the procurement cost is zero, resulting in a zero-production cost.

$$C_{Production,i} = N_{sys} PC_i \tag{4.7}$$

[9] The GLCC model implicitly assumes that the total development cost is the linear summation of all component development costs, as is the case for production costs. However, it does not account for the benefits of component reuse.

4.4.3 Recurring Cost ($Cost_{R,k,i}$)

According to Equations (4.3) and (4.5), determining the recurring cost of a component or interface involves two key factors: the timing of these recurring costs and the monetary value associated with each one. The timing of recurring costs is contingent upon the refresh schedule of the component or interface, which is determined by various factors, including the predetermined input refresh review interval (T), the support life of the component or interface (O), and the system architecture's topology. Once the times of the refresh points are determined, the discount factors ($f_{k,i}$) in Equation (4.3) can be evaluated, which are used to convert the recurring cost to present value at year zero. The specific methodology for evaluating these refresh schedules for each component and interface is elaborated upon in subsequent sections.

The amount of recurring cost incurred in each refresh cycle is based on the four distinct elements, refresh, maintenance, bridge buy, and holding. Figure 4.3 presents a cash flow diagram depicting a typical refresh cycle for a component. In this example, the refresh cycle commences at T_0 and spans a duration of T years, that is, T is the time until the next refresh. When the component is still supportable (not obsolete yet), the components are procured from the market for failure maintenance annually. The component reaches obsolescence at T_0+O. During the interval from T_0 to T_0+O, annual maintenance costs are incurred to address component failures. At the point of component obsolescence (T_0+O), a bridge buy operation is executed to secure spare parts, ensuring continued support for the component until the subsequent refresh. From T_0+O to T_0+T, holding costs are assessed annually. These costs progressively decrease each year, as the spares in inventory diminishes. Finally, at the conclusion of the refresh cycle, the refresh cost is incurred for the development and production of the next version of the component.

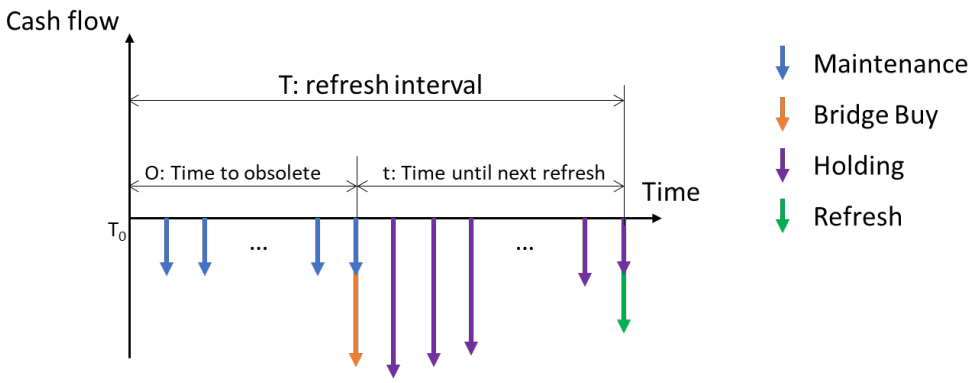


Figure 4.3 The recurring cost cash flow diagram within a refresh cycle.

To derive the four recurring cost elements, i.e., calculating $C_{Refresh,k,i}$, $C_{Maintenance,k,i}$, $C_{BridgeBuy,k,i}$, and $C_{Hold,k,i}$, the cash flows presented in Figure 4.3 should all be compounded to the end of the refresh cycle (later, this result will be discounted back to time 0).

4.4.4 Determination of the Refresh Schedule

In the GLCC model, determining the refresh schedule for individual components and interfaces holds immense significance. To simplify, the determination of refresh points for each component and interface is the same as evaluating the number of components and interfaces to refresh in every refresh review interval (T). However, this determination presents challenges. It is not solely dependent on the support lives (O) of components and interfaces but is also influenced by the architecture's topology. The architecture's topology can introduce a "ripple effect," wherein a component that remains supportable needs refreshing because it is linked to an obsolete component with a closed interface. Even though the GLCC model assumes that only obsolete components/interfaces require refresh, the ripple effect may result in refreshing more components/interfaces during each refresh review cycle than the number of obsolete components/interfaces. During the refresh review, an examination of the architecture takes place to determine whether components/interfaces require refreshing. The rules for refreshing can be summarized as follows:

1. Obsolete components must be refreshed.
2. If a component is refreshed, the connected interfaces only need to be refreshed if they are also obsolete.
3. If an interface is refreshed, the connected components on both ends of the interface must be refreshed.

According to Rule 1, obsolete components are refreshed to ensure their supportability. The GLCC model assumes that there are no mandatory refreshes due to performance or capability upgrades for the components. Rule 2 states that if the interface is still supported (i.e., not obsolete) when a component is refreshed, a "plug-and-play" approach can be implemented. This means that we can just select or redevelop a component that conforms with the interface, without making changes to the interface itself. However, if the interface is already obsolete when a component is refreshed, the interface must be refreshed as well. Rule 3 states that when an interface is refreshed, the component on both sides of the interface also needs to be refreshed to ensure compatibility with the updated interface.

Based on these three fundamental refresh rules, the refresh of a component can trigger a ripple effect throughout the entire architecture, i.e., a chain reaction, resulting in more components requiring refreshment, even if they are not yet obsolete.

Typically, an open interface has a longer support life, making it easier to exercise "plug-and-play" as long as there are components available that are compatible with the interface standard. Conversely, a closed interface is specifically designed for the connected components. As long as the connected components are available, no changes are required for the closed interface. However, if any connected component becomes obsolete, the closed interface must be refreshed. In other words, the closed interface has a support life of zero.

To capture the ripple effect of refreshes in each cycle, a series of network analyses is

conducted. The analysis follows the steps outlined below.

Step 1: Creating Sub-networks

The original architecture network is divided into multiple sub-networks by removing the edges of the open interfaces. These sub-networks can consist of combinations of components and interfaces, or they may be comprised of just a single component.

Step 2: Refreshing Components in the Same Sub-networks

Components within the same sub-networks are refreshed together since they are interconnected through closed interfaces. When the component with the shortest support life becomes obsolete, all components and close interfaces within the sub-network are refreshed simultaneously.

Step 3: Ripple Effect from Interface Refresh

In addition to component refresh, the refresh of open interfaces also influences the architecture. Each open interface connects one or two sub-networks. According to Rule 3, when an open interface is refreshed, the connected components on both ends of the interface must be refreshed. With Rule 2, this further triggers the refresh of the two (or one) sub-networks associated with the interface.

By considering these refresh rules and analyzing the network, the dates of the refresh points of the components and interfaces can be determined. In Section 4.2.6, a numerical example will be presented to demonstrate how the ripple effect and refresh dates are evaluated.

4.4.5 The Monetary Value of the Recurring Cost ($Cost_{R,k,i}$)

Table 4.2 serves as a summary of the cash flow diagram depicted in Figure 4.3, offering analytical representations for each cash flow element. Within Table 4.2, a breakdown of the financial expenditures occurring annually is provided, covering the entire duration of a refresh

cycle, commencing from T_0 and concluding at $T_0 + T$.[10] The first two columns delineate the timeline, initiating at the commencement of the refresh cycle, denoted as T_0. The component remains in operation until it becomes obsolete at $T_0 + O$, where O represents the time to component obsolescence. Following obsolescence, the component continues to receive support for t years (t=T-O) until it reaches its subsequent refresh point.

Table 4.2 The Monetary Values of Recurring Cost at Each Year During a Refresh Cycle.

	Time (year)	Description	Cash flow			
			Refresh	Maintenance	Bridge Buy	Holding
k^{th} refresh cycle of component i	T_0	Beginning of a refresh cycle	-	-	-	-
	T_0+1			$N_{sys}\lambda_i PC_i$		
	T_0+2			$N_{sys}\lambda_i PC_i$		
	…			…		
	T_0+O-1			$N_{sys}\lambda_i PC_i$		
	T_0+O	Component obsolete	-	$N_{sys}\lambda_i PC_i$	$(BN_{sys}\lambda_i t)PC_i$	
	T_0+O+1		-	-	-	$(BN_{sys}\lambda_i t)C_{h,i}$
	T_0+O+2					$(BN_{sys}\lambda_i t-N_{sys}\lambda_i)C_{h,i}$
	T_0+O+3					$(BN_{sys}\lambda_i t-2N_{sys}\lambda_i)C_{h,i}$
	…					…
	T_0+O+(t-1) =T_0+T-1					$(BN_{sys}\lambda_i t-(t-2)N_{sys}\lambda_i)C_{h,i}$
	T_0+O+t =T_0+T	End of a refresh cycle	$DC_i+N_{sys}PC_i$			$(BN_{sys}\lambda_i t-(t-3)N_{sys}\lambda_i)C_{h,i}$

The columns following the third column in Table 4.2 detail the recurring subcategory costs, providing both their monetary values and the years in which these costs are incurred. It is crucial to adjust all these values by compounding them to the end of the refresh cycle, denoted as $T_0 + T$. This adjustment is necessary to align these costs with those referenced in Equation (4.5).

Table 4.2 offers an illustration of a scenario in which a component becomes obsolete in the middle of a refresh cycle. However, it is also plausible that a component might remain non-

[10] Note that T_0 in this context symbolizes the commencement of any refresh cycle across the timeline, while T denotes the time span from T_0 to the subsequent refresh. T_0 doesn't denote a particular moment on the timeline. To illustrate with Figure 4.2, T_0 might refer to instances like time at 0, the 1st refresh, the 2nd refresh, and so forth. Consequently, T represents the duration between [0, 1st refresh], [1st refresh, 2nd refresh], [2nd refresh, 3rd refresh], and so forth.

obsolete throughout its refresh cycle, only to be compelled to undergo a refresh at the refresh cycle's conclusion for reasons such as ripple effect. In such instances, no bridge buy or holding cost would be associated with this component during the refresh cycle.

4.4.6 Recurring Refresh Costs ($C_{Refresh,k,i}$)

The third column in Table 4.2 represents the refresh cost in the refresh cycle. In the real world, the refresh of systems includes the development of the architecture and the deployment of the refresh design, i.e., install the new architecture into the fielded systems. The development and the deployment of a refresh could take several years to complete. The development cycle and deployment cycles can be different as well. This characteristic was considered in the simulation, but is difficult to include in an analytic function. In the GLCC model, both development and deployment occur at the same time and the corresponding costs are charged at the end of the refresh cycle.

The refresh cost of a component or interface i is shown in Equation (4.8), which is the development cost plus the production cost, i.e., the multiplication of number of system and procurement cost, of the component/interface. Note that for an interface, there would be no production cost.

$$C_{Refresh,k,i} = DC_i + N_{sys}PC_i \tag{4.8}$$

4.4.7 Recurring Maintenance Costs ($C_{Maintenance,k,i}$)

From the 4th to the 6th column in Table 4.2, the three other cost factors: maintenance, bridge buy, and holding costs are addressed, which pertain exclusively to components within the system. Interfaces are not subject to these costs, as they do not involve physical purchases, installations, or holding considerations. Consequently, these cost factors are applicable solely to the components' life cycle and do not extend to the interfaces within the system.

Maintenance is the process of restoring a system to its original performance with inspection, failure diagnosis, testing and part repair or replacement. The total maintenance cost is proportional to the maintenance actions performed on the systems. In this study, all components are assumed to fail independently. Only corrective maintenance is assumed. No preventative or condition-based maintenance is considered. When components fail, no repair can be done on the failed components. The failed components are replaced with good-as-new components. All procurements occur right after a part failure.

Based on these assumptions, the total number of maintenance events is the same as the total number of component failures. During each maintenance event, there is a process of inspection, diagnosis, failed part removal, new part procurement and installation. It is assumed that the efforts to inspect, diagnose, remove and install the failed parts during maintenance are also equivalent. The costs for these efforts are omitted from the cost model (since they are assumed to be the same for all cases and subtract out of the cost differences). As a result, the maintenance cost per maintenance event can be simplified to the procurement cost of the component.

Based on the above description, the annual maintenance cost is the number of failures per year times the component procurement cost. That is, the annual maintenance cost is the product of number of system (N_{sys}), component failure rate, (λ_i) and the component procurement cost (PC_i).

$$Annual\ Maintenance\ Cost = N_{sys}\lambda_i PC_i \tag{4.9}$$

The total maintenance cost of component i is the sum of the discounted annual maintenance cost from year T_0+1 to T_0+O, the present value at T_0 of the recurring maintenance cost can be presented as Equation (4.10), where r is the discount rate.

$$Recurring\ Maintenance\ Cost_{@T_0}$$

$$= \frac{(N_{sys}\lambda_i PC_i)}{(1+r)} + \frac{(N_{sys}\lambda_i PC_i)}{(1+r)^2} + \cdots + \frac{(N_{sys}\lambda_i PC_i)}{(1+r)^O} \tag{4.10}$$

Since Equation (4.10) presents a uniform series cash flow, the present value at T_0 of the maintenance cost can be written as Equation (4.10).

$$Recurring\ Maintenance\ Cost_{@T_0} = N_{sys}\lambda_i PC_i \frac{(1+r)^O - 1}{r(1+r)^O} \tag{4.11}$$

To get the recurring maintenance cost ($C_{Maintenance,k,i}$), it has to be compounded to the future cost at the end of the refresh cycle, T_0+T,

$$C_{Maintenance,k,i} = Recurring\ Maintenance\ Cost_{@T_0+T}$$

$$= N_{sys}\lambda_i PC_i \frac{(1+r)^O - 1}{r(1+r)^O}(1+r)^T \tag{4.12}$$

$$= N_{sys}\lambda_i PC_i \frac{(1+r)^T - (1+r)^{T-O}}{r}$$

4.4.8 Recurring Bridge Buy Costs ($C_{BridgeBuy,k,i}$)

Even though the systems aim to procure a new component right after there is a component failure, this may not be practical after the component obsoletes. When a part is discontinued, it may not be possible to procure the part from an authorized source. To mitigate this issue, a bridge buy is performed right before the obsolescence. Bridge buy is the action of purchasing a specific number of components, which should be able to cover the demands from the component failure during the time between component discontinuance and the next refresh. The purchased quantity is the expected number of failures during the time until next refresh multiplied by the bridge buy ratio (B). The value of the bridge buy ratio B is generally larger than 1[11], i.e., the actual quantity of components for bridge buy is larger than the expected

[11] Note, it can also be less than 1 for a myriad of reasons ranging from budget and contractual limitations to the availability of parts from the OEM.

demands for covering potential loss during holding and the uncertainty in the spare demands.[12]

Therefore, the bridge buy cost is the purchase amount times the procurement cost of the component, as illustrated in Equation (4.12). $BN_{sys}\lambda_i t$ is the total purchase quantity of component i, where λ_i is the failure rate (number of failures per year) and t is the time between the component obsolescence and the next refresh.

$$Recurring\ Bridge\ Buy\ Cost_{@T_0+O} = (BN_{sys}\lambda_i t)PC_i \tag{4.13}$$

To get the recurring Bridge Buy cost ($C_{BridgeBuy,k,i}$), it has to be mapped to the future cost at the end of the refresh cycle, T_0+T,

$$\begin{aligned} C_{BridgeBuy,k,i} &= Recurring\ Bridge\ Buy\ Cost_{@T_0+T} \\ &= (BN_{sys}\lambda_i t)PC_i(1+r)^{(T-O)} \end{aligned} \tag{4.14}$$

where r is the discount rate.

4.4.9 Recurring Holding Costs ($C_{Hold,k,i}$)

After the bridge buy is performed, the procured components are stored as inventory until they are used or the next refresh when they are no longer useful to the systems, whichever comes first. Each component in the inventory incurs an additional inventory cost that depends on the duration of its storage. The shelf lives of the components are assumed to be longer than the duration between refreshes. In the GLCC model, the holding cost is charged annually. At the end of each year, the number of components that were stored during the year is examined. The holding cost of the period is the number of stored components multiplied by the length of period and the unit holding cost, the component procurement cost times the holding cost ratio.

The last column in Table 4.2 shows the holding cost. During the first year after the component obsolescence, between T_0+O and T_0+O+1, the inventory holds the total amount of

[12] The risk of an underbuy (buying too few components) is not addressed in the GLCC model.

the spare components purchased on bridge buy for one year, i.e., $BN_{sys}\lambda_i t$ spares. Therefore, the holding cost charged at the end of year T_0+O+1 is equal to $(BN_{sys}\lambda_i t)C_h PC_i$, where C_h is the unit holding cost ratio. Every year after, a certain number of spares are used for maintenance, and the number of spares in the inventory is reduced by $N_{sys}\lambda_i$ spares every year until the next refresh.

The total holding cost at T_0+O can be presented as in Equation (4.15) with r as the discount rate, which is the sum of the discounted annual holding cost from year T_0+O+1 to year T_0+O+t ($=T_0+T$).

$$\begin{aligned} &Recurring\ Holding\ Cost_{@T_0+O} \\ &= \frac{\left((BN_{sys}\lambda_i t)C_h PC_i\right)}{(1+r)} + \frac{\left((BN_{sys}\lambda_i t - N_{sys}\lambda_i)C_h PC_i\right)}{(1+r)^2} \\ &+ \frac{\left((BN_{sys}\lambda_i t - 2N_{sys}\lambda_i)C_h PC_i\right)}{(1+r)^3} \ldots \\ &+ \frac{\left((BN_{sys}\lambda_i t - (t-1) * N_{sys}\lambda_i)C_h PC_i\right)}{(1+r)^t} \end{aligned} \tag{4.15}$$

This cash flow is also known as the arithmetic gradient series. Equation (4.15) and (4.16) rewrite the holding cost as the present value at T_0+O, where A is equal to $(BN_{sys}\ \lambda_i t)\ C_h PC_i$, G is equal to $-(N_{sys}\lambda_i)C_h PC_i$.

$$Recurring\ Holding\ Cost_{@T_0+O} = A(P/A, r, t) + G(P/G, r, t) \tag{4.16}$$

$$Recurring\ Holding\ Cost_{@T_0+O} = A\frac{(1+r)^t - 1}{r(1+r)^t} + G\frac{(1+r)^t - rt - 1}{r^2(1+r)^t} \tag{4.17}$$

To get the recurring holding cost ($C_{Holding,k,i}$), it has to be mapped to the future cost at the end of the refresh cycle, T_0+T (or T_0+O+t),

$$C_{Holding,k,i} = Recurring\ Holding\ Cost_{@T_0+O+t}$$
$$= A\frac{(1+r)^t - 1}{r} + G\frac{(1+r)^t - rt - 1}{r^2} \tag{4.18}$$

Substitute the original values of the A and G, the recurring holding cost can be written as,

$$C_{Holding,k,i} = N_{sys}\lambda_i C_h PC_i\left(Bt\frac{(1+r)^t - 1}{r} - \frac{(1+r)^t - rt - 1}{r^2}\right) \tag{4.19}$$

Table 4.3 summarizes the analytical formulations within the GLCC model, spanning from Equations (4.1) to (4.19). The life-cycle cost (LCC) is the present value at year zero. The non-recurring cost ($Cost_{NRE,i}$) is also the present value at year zero. The recurring cost ($Cost_{R,k,i}$,) and the corresponding elements, i.e., $C_{Refresh,k,i}$, $C_{Maintenance,k,i}$, $C_{BridgeBuy,k,i}$, and $C_{Hold,k,i}$, are the future values and the end of the kth refresh cycle. Therefore, these future values need to be discounted with a discount factor ($f_{k,i}$) back to year zero.

Table 4.3 The Conclusion of the GLCC Model.

$LCC = \sum_{i=1}^{n_c+n_I} LCC_i$	
$LCC_i = Cost_{NRE,i} + \sum_{k=1}^{N_{R,i}} f_{k,i} Cost_{R,k,i}$	
$Cost_{NRE,i} = C_{Development,i} + C_{Production,i}$	
$C_{Development,i}$	DC_i
$C_{Production,i}$	$N_{sys}PC_i$
$Cost_{R,k,i} = C_{Refresh,k,i} + C_{Maintenance,k,i} + C_{BridgeBuy,k,i} + C_{Hold,k,i}$	
$C_{Refresh,k,i}$	$DC_i + N_{sys}PC_i$
$C_{Maintenance,k,i}$	$N_{sys}\lambda_i PC_i \frac{(1+r)^{T_{k,i}} - (1+r)^{T_{k,i}-O_i}}{r}$
$C_{BridgeBuy,k,i}$	$(B * N_{sys}\lambda_i t_{k,i})PC_i(1+r)^{t_{k,i}}$
$C_{Hold,k,i}$	$N_{sys}\lambda_i C_h PC_i\left(Bt_{k,i}\frac{(1+r)^{t_{k,i}} - 1}{r} - \frac{(1+r)^{t_{k,i}} - rt_{k,i} - 1}{r^2}\right)$

It's important to observe that Table 4.3 introduces slight variations in notation. Specifically, in this table, the refresh intervals (T) and the duration between component obsolescence and the subsequent refresh (t) are presented in a generalized manner, incorporating subscripts denoting the kth refresh cycle and the ith component (k and i). Similarly, the time to obsolescence (O) is also generalized, featuring subscripts corresponding to the ith component (i).

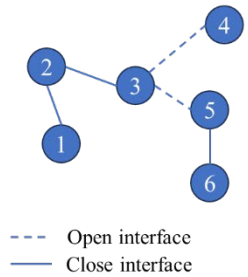
4.5 The GLCC Analysis Process

In this subsection, a numerical example is used to demonstrate the analysis process of the GLCC model. The analysis process of the GLCC model involves two steps: the determination of refresh schedule and the subsequent cost calculations.

The initial step of the process requires inputting the system's architecture network and the support lives (O) of components and interfaces. Through a network analysis that accounts for the refresh ripple effect, the refresh dates of components and interfaces within the architecture can both be determined. The concept of the ripple effect has been explained in the previous section and is important for this step of the process. Once the refresh dates are determined, the costs of each component/interface throughout the life cycle can be computed using Equations (4.6) to (4.18). In the rest of this section, a numerical example is provided, showing how the GLCC model adeptly determines the life-cycle cost of a given system.

Figure 4.4 presents the inputs of the example system.

	Support life (years) (O_i)	Development cost (DC_i)	Procurement cost (PC_i)
Component 1	3.5	DC_1	PC_1
Component 2	7.6	DC_2	PC_2
Component 3	5.2	DC_3	PC_3
Component 4	8.1	DC_4	PC_4
Component 5	6.8	DC_5	PC_5
Component 6	1.2	DC_6	PC_6



	Support life (O_i)	Development cost (DC_i)
Interface 1-2	~0	DC_7
Interface 2-3	~0	DC_8
Interface 3-4	13.5	DC_9
Interface 3-5	18.7	DC_{10}
Interface 5-6	~0	DC_{11}

Number of system (N_{sys})	20
Holding cost ratio (C_h)	0.2
Discount rate (r)	9%
Failure rate (λ)	4.6
Purchase buffer (B)	150%
End of support year (EOS)	53
Refresh review interval (T)	5

Figure 4.4 The inputs for the example system.

Figure 4.5 shows how the ripple effect is evaluated. The left plot represents the original architecture network, consisting of six components and five interfaces. Out of these interfaces, two are closed, while three are open. Following step 1 and step 2 in the ripple effect section, the original network can be divided into three sub-networks by removing the open interfaces, as shown in the middle plot. The open interfaces serve as the boundary for ripple effects. A component's refresh only triggers other components' refreshes within the same sub-network and does not extend beyond the open interfaces. These sub-networks include the following components: [1,2,3], [5,6], and [4]. Whenever any component within a sub-network becomes obsolete, all the components in that sub-network are refreshed together. This simultaneous refresh applies to all the components within the same sub-network.

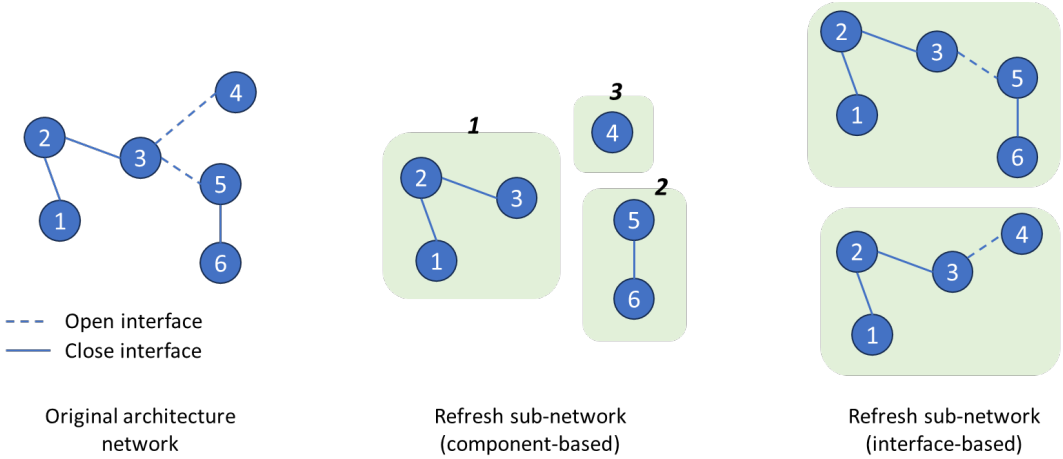


Figure 4.5 Demonstration of the ripple effect.

Additionally, the right plot in Figure 4.5 illustrates the refresh ripple effect resulting from the interfaces. For instance, interface 3-5 connects sub-networks 1 and 2. When interface 3-5 becomes obsolete and necessitates a refresh, the components in both sub-networks (i.e., [1,2,3,5,6]) must also be refreshed, along with interface 3-5.

Following the example in Figure 4.5, given the support life of each component and interface, the corresponding refresh dates can be evaluated. The values of the support life are presented in Table 4.4. Note that the support lives of the closed interfaces are set to be 0.

Table 4.5 shows the refresh dates of the components and the interfaces. For instance, components 1, 2 and 3 are in the same sub-network therefore the three components are refreshed simultaneously. The component-based refresh is on every five years since the least support life is component 1, i.e., 3.5 years. In addition, the refreshes of this sub-network also are affected by the refresh of the open interfaces 3-4 and 3-5. In short, the components and interfaces in this sub-network are refreshed every 15 and 20 years. In between the interface-based refreshes, every five years, they also have to be refreshed.

Table 4.4 The Values of the Support Lives for the Example Case

Refresh cycle	5		
	Support life		Support life
Component 1	3.5	Interface 1-2	~0
Component 2	7.6	Interface 2-3	~0
Component 3	5.2	Interface 3-4	13.5
Component 4	8.1	Interface 3-5	18.7
Component 5	6.8	Interface 5-6	~0
Component 6	1.2		

unit: years

Table 4.5 The Result of Refresh Dates for the Example Case

	Refresh cycle (component)	Refresh cycle (interface 3-4)	Refresh cycle (interface 3-5)	Refresh time
Component 1	5	15	20	5,10,**15**,**20**,25,**30**,35,**40**,**45**,50,55,**60**...
Component 2				
Component 3				
Interface 1-2				
Interface 2-3				
Component 4	10	15	-	10,**15**,25,**30**,40,**45**,55,**60**...
Component 5	5	-	20	5,10,15,**20**,25,30,35,**40**,45,50,55,**60**...
Component 6				
Interface 5-6				
Interface 3-4	-	15	-	15,30,45,60...
Interface 3-5	-	-	20	20,40,60...

unit: years

Once the refresh dates for each component and interface are determined, the next step involves computing the costs. Table 4.6 serves as a cost table, using component 1 as an example to demonstrate the computation process. The first row of the table represents year zero plus the refresh dates of component 1. With the refresh dates, the discount factors of each refresh cycle can also be calculated. Utilizing the refresh date and the component's support life (3.5 years), enables the calculation of the time between refreshes ($T_{k,1}$) and the time between obsolescence and the next refresh ($t_{k,1}$), presented in the second and third rows, respectively. These intermediate parameters play a crucial role in computing the subsequent costs.

In the subsequent step, Equations (4.6) to (4.18) are employed to fill out the remaining entries in the cost table. It's important to note that non-recurring costs are only charged in year zero, while recurring costs are charged on the refresh dates. This table provides a clear

overview of the time and value associated with each type of cost for component 1.

Table 4.6 The Cost Table for Component 1

Time	0	5	10	15	20	25	…
Time between refresh $(T_{k,1})$	-	5	5	5	5	5	…
Time till next refresh $(t_{k,1})$	-	1.5	1.5	1.5	1.5	1.5	
Discount factor$(f_{k,1})$	1	$1/(1+r)^5$	$1/(1+r)^{10}$	$1/(1+r)^{15}$	$1/(1+r)^{20}$	$1/(1+r)^{25}$	
Development $(C_{Development,1})$	DC_1	-	-	-	-	-	
Production $(C_{Production,1})$	$N_{sys}PC_1$	-	-	-	-	-	
Refresh $(C_{Refresh,k,1})$	-	$DC_1+N_{sys}PC_1$	$DC_1+N_{sys}PC_1$	$DC_1+N_{sys}PC_1$	$DC_1+N_{sys}PC_1$	$DC_1+N_{sys}PC_1$	
Maintenance $(C_{Maintenance,k,1})$	-	$N_{sys}\lambda PC_1\frac{(1+r)^5-(1+r)^{1.5}}{r}$	$N_{sys}\lambda PC_1\frac{(1+r)^5-(1+r)^{1.5}}{r}$	$N_{sys}\lambda PC_1\frac{(1+r)^5-(1+r)^{1.5}}{r}$	$N_{sys}\lambda PC_1\frac{(1+r)^5-(1+r)^{1.5}}{r}$	$N_{sys}\lambda PC_1\frac{(1+r)^5-(1+r)^{1.5}}{r}$	
Bridge buy $(C_{BridgeBuy,k,1})$	-	$(B*N_{sys}\lambda*1.5)PC_1(1+r)^{1.5}$	$(B*N_{sys}\lambda*1.5)PC_1(1+r)^{1.5}$	$(B*N_{sys}\lambda*1.5)PC_1(1+r)^{1.5}$	$(B*N_{sys}\lambda*1.5)PC_1(1+r)^{1.5}$	$(B*N_{sys}\lambda*1.5)PC_1(1+r)^{1.5}$	
Holding $(C_{Hold,k,1})$	-	$N_{sys}\lambda C_h\left(Bt\frac{(1+r)^{1.5}-1}{r}-\frac{(1+r)^{1.5}-1.5r-1}{r^2}\right)$	$N_{sys}\lambda C_h\left(Bt\frac{(1+r)^{1.5}-1}{r}-\frac{(1+r)^{1.5}-1.5r-1}{r^2}\right)$	$N_{sys}\lambda C_h\left(Bt\frac{(1+r)^{1.5}-1}{r}-\frac{(1+r)^{1.5}-1.5r-1}{r^2}\right)$	$N_{sys}\lambda C_h\left(Bt\frac{(1+r)^{1.5}-1}{r}-\frac{(1+r)^{1.5}-1.5r-1}{r^2}\right)$	$N_{sys}\lambda C_h\left(Bt\frac{(1+r)^{1.5}-1}{r}-\frac{(1+r)^{1.5}-1.5r-1}{r^2}\right)$	

To determine the total life-cycle cost for component 1, the costs are discounted based on their corresponding charging times and then summed up. Similar cost tables can be constructed for all other components and interfaces within the system. Once the cost tables are completed, the total life-cycle cost can be evaluated by considering the costs associated with all components and interfaces. The life-cycle cost computed here is not the actual cost. Lots of "wash" costs are removed from the cost model. The value of the life-cycle cost is relative for selecting the optimal openness, i.e., the cost differences between system implementations are valid.

4.6 A-RCI Architecture Simplification

The previous section outlined the development of the Generalized Life Cycle Cost (GLCC) model. Validation is now required to substantiate the practicality of this model. This validation procedure centers on a comparison between the life-cycle cost difference projections generated by the GLCC model and those obtained from the simulation discussed in Chapter 3. For the validation process, the data in Table 3.1, sourced from the A-RCI, was used as inputs for both

the GLCC model and the simulation.

In order to clearly demonstrate the GLCC model, a simplified version of the A-RCI architecture network is derived. Instead of utilizing all components and interfaces from the original architecture, the simplified A-RCI architecture integrates all the components and interfaces into a few essential modules, reducing complexity while retaining the core system attributes. This streamlined A-RCI architecture enables the GLCC model to conduct more extensive analyses.

In constructing the simplified A-RCI architecture, the initial step involved extracting a simplified system network derived from the original A-RCI architecture network. This was achieved by employing module nodes to represent clusters of components found within the original system network. Subsequently, the necessary parameters including development costs, procurement costs, and reliability for these module nodes within the simplified network can be computed. Finally, to validate the effectiveness of the simplified architecture, a comparison between the life-cycle cost outcomes generated by the simplified architecture and those of the original architecture were compared.

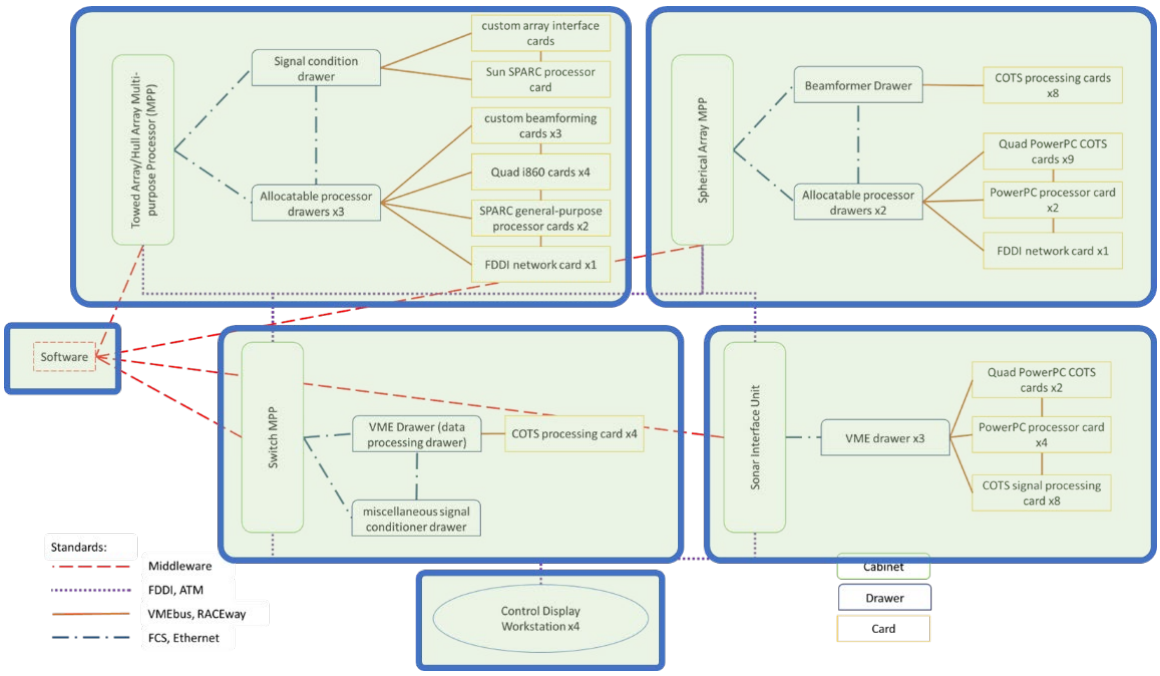


Figure 4.6 Simplification of the original A-RCI architecture.

Figure 4.6 illustrates the original A-RCI architecture and the process by which the simplified architecture network was derived. The green boxes with blue outlines are the modules nodes to represent clusters of components and interfaces. The original A-RCI architecture network comprises 147 components and 1330 interfaces. While it was technically feasible to include the entire network in the GLCC model, the computational time required for validation was deemed excessively lengthy. Consequently, a methodical assessment of the complete A-RCI architecture network to create a simplified version that would provide a representative snapshot of the full architecture's capabilities was performed.

Within this architecture network, the A-RCI is composed of software, a control station, and four additional modules. These elements are interconnected using either middleware or the ATM standard interface. This simplified architecture serves as a foundational reference point for subsequent validation of the GLCC model.

Once the simplified architecture was established, the subsequent phase involves determining the characteristic parameters pertinent to the modules within this simplified structure. These parameters serve as inputs for both the GLCC model and the simulation, facilitating later validation analysis of the GLCC model. The module characteristic parameters include development cost (DC), procurement cost (PC), and failure rate (λ), which serves as the inputs for the GLCC model.

The calculations for these module parameters are outlined in Equations (4.20) to (4.22). To compute the development cost of a module, the development costs DC associated with the components and the interfaces contained within that module are summed. The procurement cost PC of a module is derived by summing the procurement costs of the components within it, it is assumed that no procurement cost is linked to the interfaces.

The failure rate λ of a module is calculated by considering the annual maintenance cost. This cost is computed as the sum of the products obtained by multiplying the failure rate with the procurement cost of each component within the module. With the annual maintenance cost and the module's procurement cost, as derived in Equation (4.11), the failure rate associated with the module can be determined.

$$DC_{module} = \Sigma DC_i \tag{4.20}$$

$$PC_{module} = \Sigma PC_i \tag{4.21}$$

$$\lambda_{module} \times PC_{module} = \Sigma \lambda_i PC_i \tag{4.22}$$

Figure 4.7 shows the inputs for the simplified A-RCI architecture based on the proposed calibration process proposed. Note that the software is assumed to have no failure rate ($\lambda = 0$).

Component i	Name	Development cost DC_i ($)	Procurement cost PC_i ($)	Failure rate (failures/year)
1	Towed Array MPP	$9,502,777	$2,285,792	0.037613631
2	Switch MPP	$6,141,667	$1,629,324	0.033949469
3	Spherical Array MPP	$6,898,611	$2,293,240	0.037710916
4	Sonar Interface Unit	$6,398,611	$1,907,902	0.045063602
5	Control Display Workstation	$1,000,000	$1,600,000	0.067567568
6	Software	$12,500,000	$90,909	-

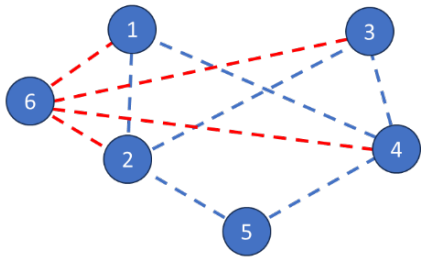


Figure 4.7 The inputs for the simplified A-RCI architecture.

Figure 4.8 displays the results of simulations for both the original A-RCI and the simplified A-RCI, each solution includes 300 timelines. It's important to note that the vertical axis, representing the life-cycle cost, doesn't directly equate to the actual monetary expenditure on A-RCI. Instead, it reflects the cost associated with the system's openness based on the model assumptions.

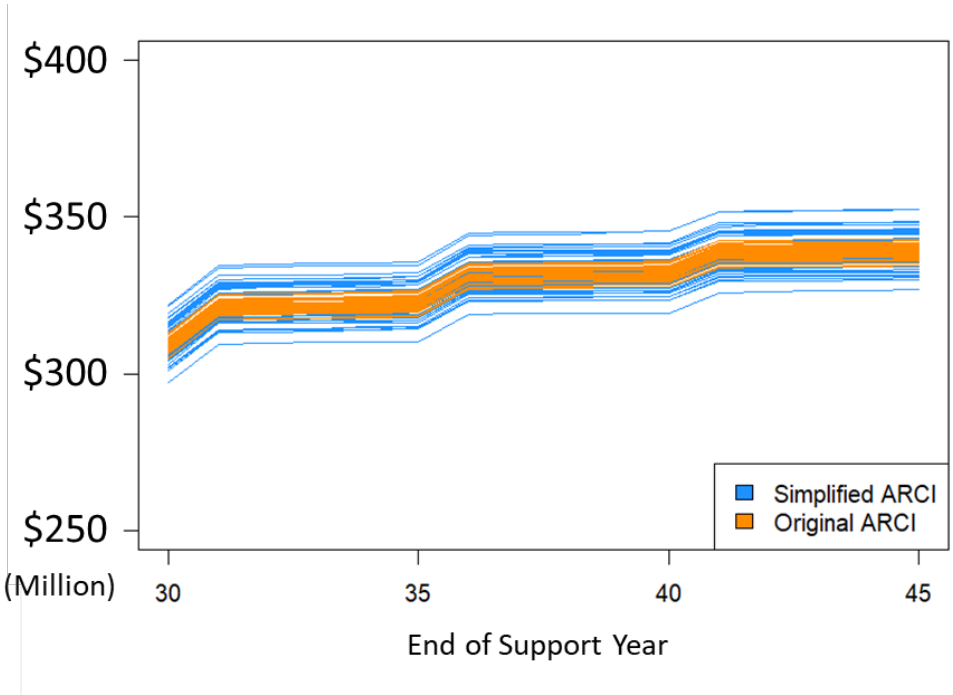


Figure 4.8 The comparison of simulations of the original A-RCI and the simplified A-RCI.

The mean values from the simulation of the full A-RCI (Chapter 3) and the simplified A-RCI exhibit similarity. However, the results for the simplified A-RCI demonstrate a higher variance. This variance is attributed to the fact that the simplified A-RCI is represented by modules, which are combinations of components, and each module incurs a higher per-unit cost. By contrast, the original A-RCI can capture the intricacies of individual component failures, whereas the simplified A-RCI cannot account for partial failures within a module. In other words, when one module fails, all the components within it are considered failed. Consequently, a greater variance in the life-cycle cost of the simplified A-RCI is observed. While the variance might not be entirely representative, the mean values of both cases are sufficiently close.

4.7 Implementing A-RCI and Validation of the GLCC model

The previous section showed that the simplified A-RCI architecture can effectively replace the original A-RCI architecture by generating similar life-cycle cost mean values. In this section, the simulation result from the simplified A-RCI architecture is used as a baseline to validate

the GLCC model. This section will show that with the same inputs, the GLCC model can produce results that are similar to the simulation.

Figure 4.9 shows the life-cycle cost comparison between the simulation of the simplified A-RCI and the calculation of the GLCC model. The inputs are the same as was used for the previous case shown in Figure 4.8. The simulation generated 300 timelines and the GLCC computed an average life-cycle cost over the end-of-support year. As discussed in the assumptions section, there were no uncertainties in the input parameters used in the GLCC model. The result shows that the GLCC model can generate a representative life-cycle cost mean value curve for the simulation.

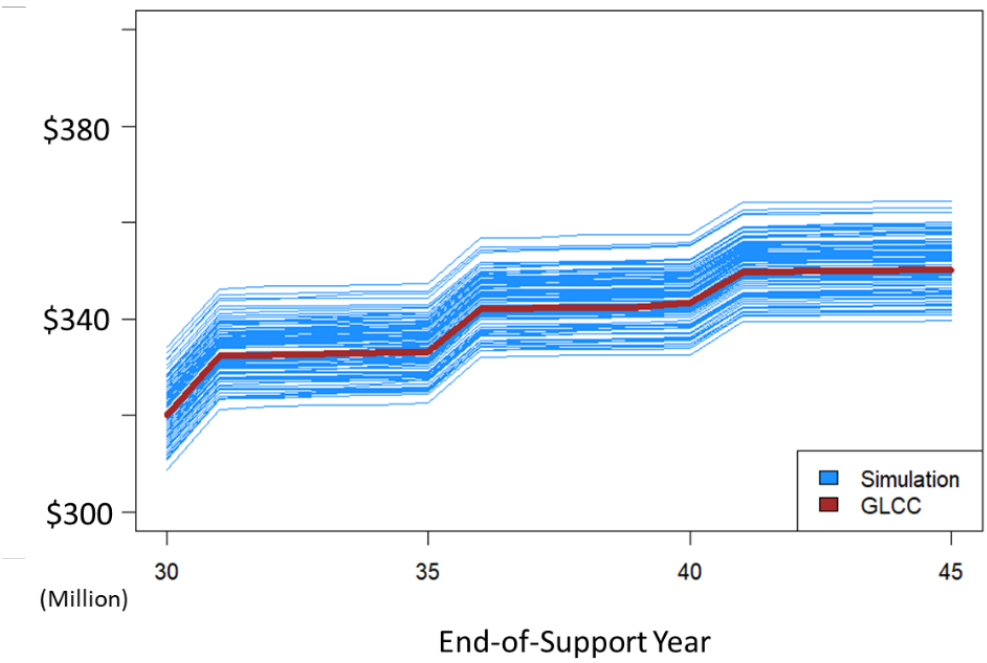


Figure 4.9 The life-cycle cost comparison between the simulation of the simplified A-RCI and the calculation of the GLCC model.

To assess the robustness of the GLCC model, various result comparisons were conducted by introducing variations in other parameters, including the discount rate and refresh review interval. Figures 4.10 and 4.11 illustrate the comparisons between the simulation and GLCC model approaches under different discount rates and refresh review intervals. The solid lines represent the results from the GLCC model, while the shaded areas depict the simulation results

generated from 100 timelines. In both figures, the GLCC model results exhibit a consistent trend with the simulation results, with the GLCC curves positioned approximately at the means of the simulation timelines. Also, note that the baseline conditions, characterized by a 7% discount rate and a 5-year refresh review interval, yield results similar to those in Figure 4.9.

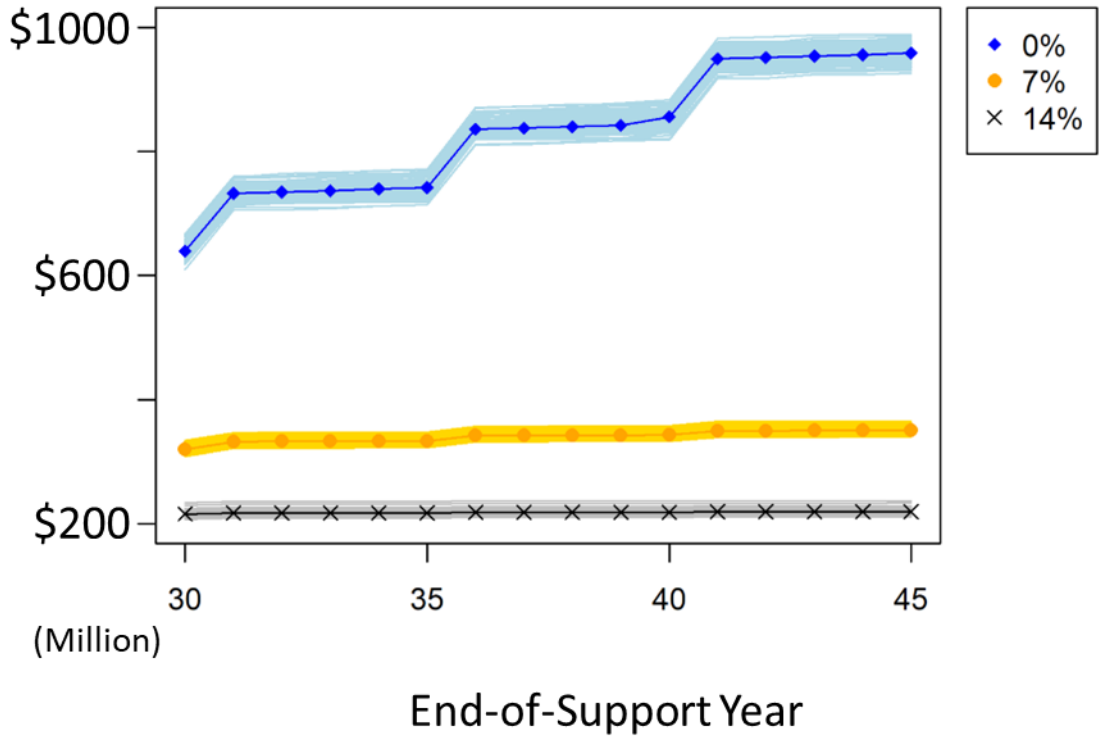


Figure 4.10 The life-cycle cost comparison between the simulation of the simplified A-RCI and the calculation of the GLCC model with varied discount rate.

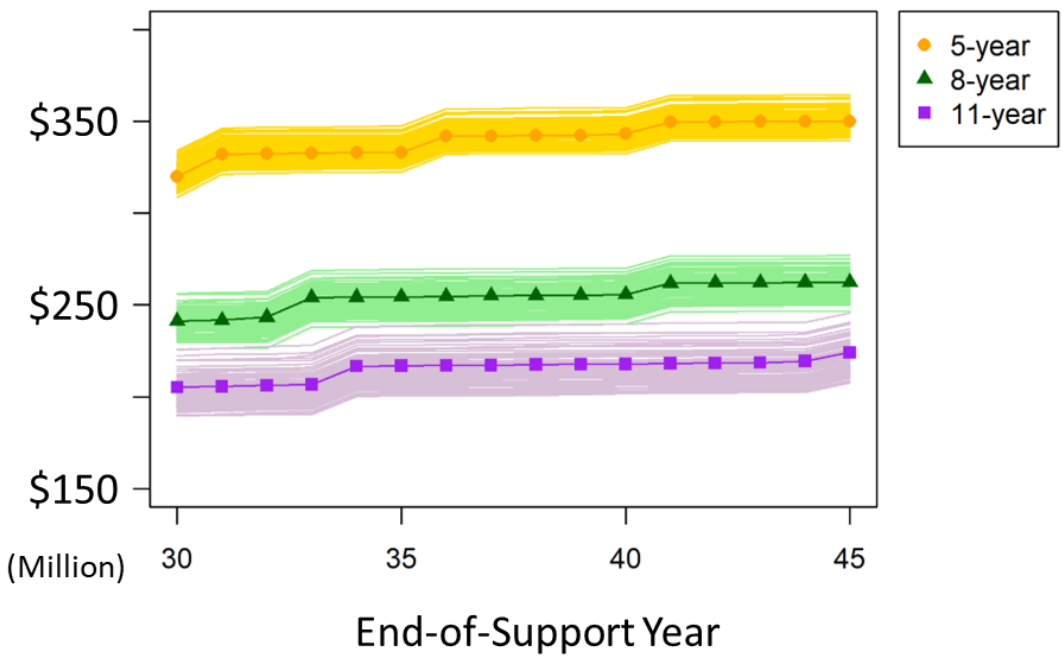


Figure 4.11 The life-cycle cost comparison between the simulation of simplified A-RCI and the calculation of the GLCC model with varied refresh review interval.

4.8 GLCC Application

This section will utilize the A-RCI case to illustrate how the GLCC model can effectively determine the optimal system openness. The increased computational speed provided by the GLCC model allows a comprehensive exploration of various openness scenarios, ultimately pinpointing the optimal level of openness that results in the most favorable life-cycle cost.

The parameters utilized in the preceding section serve as the foundational inputs for the open system baseline. The initial assumption is that all components and interfaces within the baseline A-RCI architecture are open by default. Subsequently, randomly generated cost and support life parameters for the closed components and interfaces are introduced, adhering to specific rules:

1. Closed components incur higher development and procurement costs compared to their open counterparts.

2. The support lives of closed components are longer than those of open components.
3. Closed interfaces, in contrast, have lower development costs than open interfaces.
4. The support lives of closed interfaces are assumed to be zero, meaning that when a component is redeveloped, the connected closed interfaces are also required to be redeveloped.

Following these rules, the parameters for the closed versions of components and interfaces can be generated. Note that the development and procurement costs for the closed components fall within a range that is 1-10 times that of their open counterparts. The development costs of the closed interfaces are in the range of 0.1-1 times that of their open counterparts. Figure 4.12 lists all the parameters employed in the GLCC model.

	Open			Closed			
	O_i	DC_i	PC_i	O_i	Dc_i	PC_i	λ_i
Comp1	3.00	9,502,777	2,285,792	6.12	54,364,417	19,007,963	0.0376
Comp2	3.00	6,141,667	1,629,324	8.87	40,488,329	1,663,047	0.0339
Comp3	3.00	6,898,611	2,293,240	5.34	65,089,291	10,317,992	0.0377
Comp4	3.00	6,398,611	1,907,902	6.30	21,612,223	3,156,614	0.0451
Comp5	3.00	1,000,000	1,600,000	4.37	9,994,288	13,112,612	0.0676
Comp6	3.00	12,500,000	90,909	9.45	87,311,068	776,672	0.0000

		Open		Closed	
		O_i	DC_i	O_i	DC_i
interface1	[1,2]	15	220,000	0.001	51,883
interface2	[1,4]	15	220,000	0.001	141,148
interface3	[1,6]	15	220,000	0.001	27,425
interface4	[2,3]	15	220,000	0.001	151,300
interface5	[2,5]	15	220,000	0.001	22,315
interface6	[2,6]	15	220,000	0.001	91,662
interface7	[3,4]	15	220,000	0.001	42,331
interface8	[3,6]	15	220,000	0.001	22,072
interface9	[4,5]	15	220,000	0.001	206,501
interface10	[4,6]	15	220,000	0.001	60,099

Randomly generated values

Figure 4.12 The parameters employed in the GLCC model, including the open and closed counterparts.

According to the simplified A-RCI architecture network, there are a total of 6 components and 10 interfaces, each offering two alternatives: an open or closed version. Consequently, there are a total of 2^{16} = 65,536 potential openness scenarios. Subsequently, all these scenarios are input into the GLCC model to calculate the life-cycle cost.

Figure 4.13 illustrates the life-cycle cost outcomes for all 65,536 scenarios, assuming an end-of-support year of 50 years. Although the absolute life-cycle cost values may change

across end-of-support years, the relative order of costs among scenarios remains consistent. If one scenario is more costly at an end-of-support year of 30, it also proves more expensive at an end-of-support year of 50. Thus, within the framework of the GLCC model, the end-of-support year does not impact the result of optimal openness.

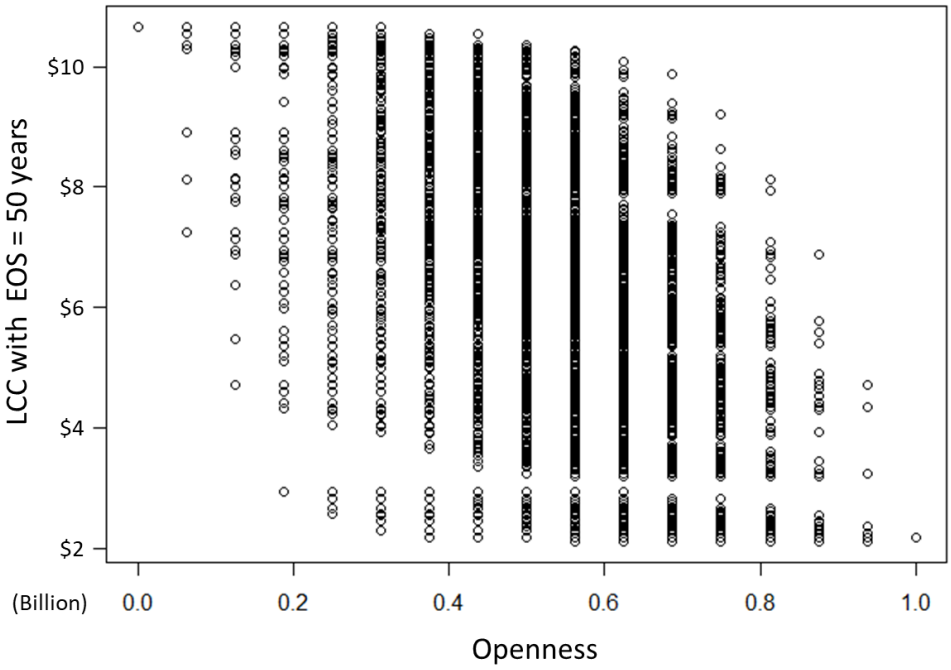


Figure 4.13 The life-cycle cost results for all 65,536 scenarios, with an end-of-support year of 50 years.

The vertical-axis in Figure 4.13 represents the relative life-cycle cost, providing a quantitative basis for comparing cost differences between scenarios. The horizontal axis represents system openness, defined as the ratio of open components and interfaces to the total number of components and interfaces in the system. Figure 4.13 reveals a general trend of decreasing life-cycle costs with increasing system openness, suggesting that, in most scenarios, a more open system tends to be more cost-effective than a less open one.

However, when focusing on the lower part of Figure 4.13, with the vertical axis ranging from 1,883 million to 1,885 million as shown in Figure 4.14, the optimal openness doesn't occur at 100% but seems to be around 69%. This result implies that opting for a fully open architecture might not yield the most cost-effective outcome.

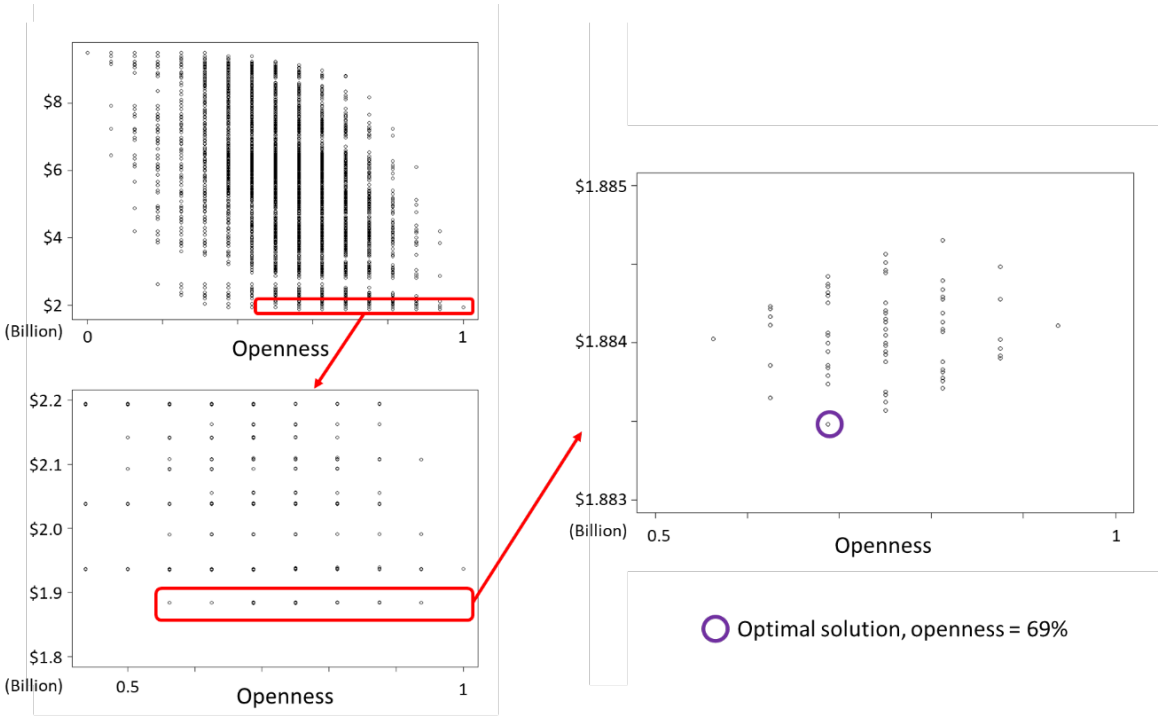


Figure 4.14 A detailed examination of the openness result.

Table 4.7 analyzes the individual components and interfaces and reveals the optimal openness selection. Except for component 2, all other five components are recommended to be open. This is possibly due to the closed version of component 2 having a much longer support life, and its development and procurement costs are not excessively high, making the closed component 2 more favorable.

Table 4.7 Optimal Openness Selection for Individual Components and Interfaces

	Open			Closed			Result
	O_i	DC_i	PC_i	O_i	Dc_i	PC_i	
Comp1	3.00	9,502,777	2,285,792	6.12	54,364,417	19,007,963	Open
Comp2	3.00	6,141,667	1,629,324	8.87	40,488,329	1,663,047	Closed
Comp3	3.00	6,898,611	2,293,240	5.34	65,089,291	10,317,992	Open
Comp4	3.00	6,398,611	1,907,902	6.30	21,612,223	3,156,614	Open
Comp5	3.00	1,000,000	1,600,000	4.37	9,994,288	13,112,612	Open
Comp6	3.00	12,500,000	90,909	9.45	87,311,068	776,672	Open

		Open		Closed		Result
		O_i	DC_i	O_i	DC_i	
interface1	[1,2]	15	220,000	0.001	51,883	Open
interface2	[1,4]	15	220,000	0.001	141,148	Open
interface3	[1,6]	15	220,000	0.001	27,425	Closed
interface4	[2,3]	15	220,000	0.001	151,300	Open
interface5	[2,5]	15	220,000	0.001	22,315	Open
interface6	[2,6]	15	220,000	0.001	91,662	Open
interface7	[3,4]	15	220,000	0.001	42,331	Closed
interface8	[3,6]	15	220,000	0.001	22,072	Closed
interface9	[4,5]	15	220,000	0.001	206,501	Open
interface10	[4,6]	15	220,000	0.001	60,099	Closed

In contrast, the interfaces exhibit more even results. Four out of ten interfaces should be closed, and the remainder should be open. The choice of closed interfaces is influenced by lower development costs, but interface 5, which is connected to components 2 and 5, is recommended to be open, even though the closed version has a much lower development cost ($22.3k). This suggests that the advantage of using an open interface goes beyond saving on development costs. Open interfaces play a critical role in preventing the ripple effect of refresh, which is challenging to quantify. Interface 5, for instance, connected to a closed component 2, can prevent the ripple effect from other components, reducing the number of refreshes for component 2.

Further exploration of optimal openness alternatives shows commonalities. The alternatives in the right plot of Figure 4.14 represent optimal solutions with a variance of less than 2 million dollars. Examining the openness configuration reveals that certain components and interfaces have the same openness selection. Component 2 needs to be closed, other components need to be open, and interfaces 1, 4, 5, and 6 must be open. The remaining interfaces can be either open

or closed. This discovery indicates critical and non-critical components and interfaces. Critical components or interfaces, sharing the same openness selection in optimal solutions, significantly impact life-cycle costs, while non-critical ones have minor influence, up to two million dollars in this case.

The scale of development/ procurement costs and the influence on the ripple effect are factors contributing to the criticality of components and interfaces. These interfaces play a crucial role in "protecting" component 2 from undergoing refreshes along with other connected open components. The expenditure associated with a single additional refresh for component 2 is estimated at approximately 86 million dollars. Utilizing these open interfaces allows component 2 to undergo the least number of refreshes, resulting in cost savings in the hundreds of millions of dollars.

The preference for more open scenarios, however, depends on various factors, including the benefits derived from both open and closed components/interfaces, as well as other parameters such as the number of systems, refresh cycle, discount rate, etc. Although the impact of these parameters on favoring a more open system is currently unknown (to be discussed in the next section), one can intuitively guess that a system would lean towards openness when the benefits of open components/interfaces outweigh those of closed ones. Given the parameter setup in this case, it is plausible that the advantages of open components/interfaces surpass those of their closed counterparts, rendering the selection of closed components or interfaces unnecessary.

To test this hypothesis, another parameter set is generated. In this set, the difference between the development/procurement costs of open and closed components is reduced, thereby weakening the advantage of open components being less costly in development and procurement. Specifically, the development/ procurement costs of closed components are set

to be 1-2 times greater than their open counterparts, instead of 1-10 times greater in previous case. This newly generated parameter set is detailed in Table 4.8.

Table 4.8 Parameter Set with Less Development/Procurement Cost Difference between Open and Closed Components

	Open			Closed		
	O_i	DC_i	PC_i	O_i	Dc_i	PC_i
Comp1	3.00	9,502,777	2,285,792	8.78	11,080,214	2,418,301
Comp2	3.00	6,141,667	1,629,324	14.88	6,643,459	1,822,020
Comp3	3.00	6,898,611	2,293,240	12.85	7,332,447	2,301,189
Comp4	3.00	6,398,611	1,907,902	7.93	7,271,541	2,152,995
Comp5	3.00	1,000,000	1,600,000	8.51	1,119,472	1,828,025
Comp6	3.00	12,500,000	90,909	10.67	13,854,559	106,983

		Open		Closed	
		O_i	DC_i	O_i	DC_i
interface1	[1,2]	15	220,000	0.001	51,883
interface2	[1,4]	15	220,000	0.001	141,148
interface3	[1,6]	15	220,000	0.001	27,425
interface4	[2,3]	15	220,000	0.001	151,300
interface5	[2,5]	15	220,000	0.001	22,315
interface6	[2,6]	15	220,000	0.001	91,662
interface7	[3,4]	15	220,000	0.001	42,331
interface8	[3,6]	15	220,000	0.001	22,072
interface9	[4,5]	15	220,000	0.001	206,501
interface10	[4,6]	15	220,000	0.001	60,099

Figure 4.15 displays the life-cycle cost outcomes for all 65,536 scenarios under the recently generated parameter sets. In contrast to the trend observed in Figure 4.13, the data points in Figure 4.15 indicate an increase in life-cycle costs as system openness expands. Consequently, for the sake of cost-effectiveness, the system should lean towards a more closed configuration. It's worth noting that the data point region in Figure 4.15 is not as complete as that in Figure 4.13, featuring a notch on the left side of the region. This notch arises from the input variations in the architecture network and the refresh cycle, and it is an inherent outcome of the GLCC model given these specific inputs.

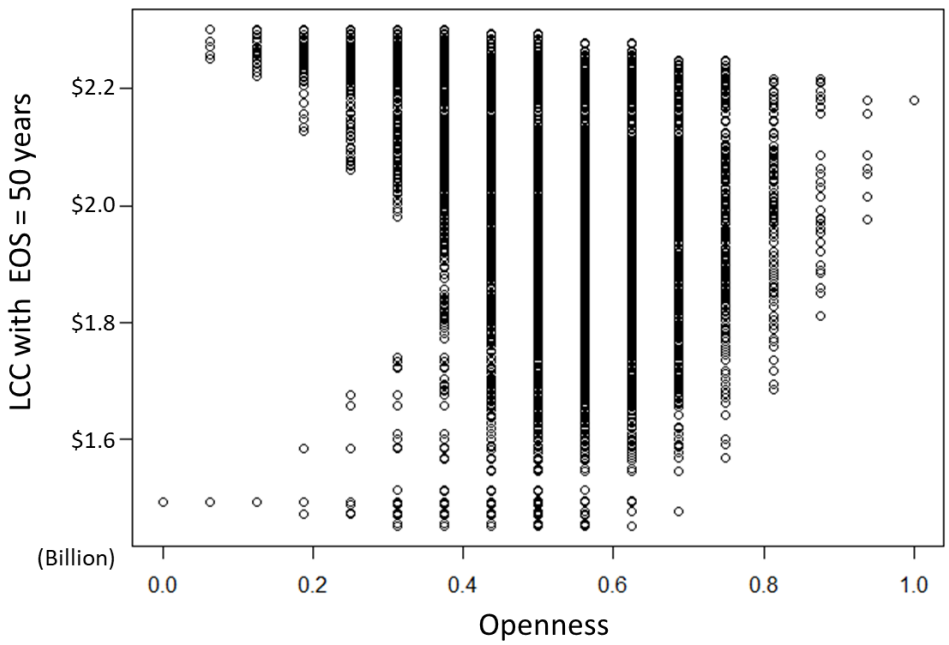


Figure 4.15 The life-cycle cost results for all 65,536 scenarios, with an end-of-support year of 50 years.

4.9 Conclusion

In summary, the GLCC model relies on simulations with specific assumptions, simplifying treatment for complex situations like varied refresh policies, production/retirement schedules, and parameter uncertainty. It evaluates the refresh interval and dependency within the architecture network, calculating the life-cycle cost of individual components and interfaces to determine the overall life-cycle cost. Although the GLCC model generates deterministic values and lacks the ability to capture uncertainty present in simulations, its rapid computation speed facilitates the exploration of various system architecture openness combinations to identify optimal configurations.

Through the implementation in A-RCI, the GLCC model has proven its ability to assess the life-cycle cost across the entire spectrum of possible openness, accomplishing this task with a quicker computational speed compared to simulation. Both cases reveal that a fully open system may not consistently be the most cost-effective choice. While the first case leans towards a preference for greater openness, the second case, under distinct parameter

configurations, suggests a potential preference for a relatively more closed system. Although these cases underscore the considerable influence of parameters in deciding whether to employ an open system, the precise mechanism by which these parameters impact the selection of system openness remains unknown.

Chapter 5 Generalized Open Systems Life-Cycle Cost Predictors

In Chapter 4, the effectiveness of the GLCC model was established, emphasizing its ability to significantly reduce computing times compared to simulations while yielding comparable results. The GLCC model can explore all possible openness and select the openness alternative with the minimum computed life-cycle cost. In the previous chapter, the demonstration results showed how different parameter setups, to be more specific, the development and procurement costs, led to different preferred openness. However, how the parameters affect the openness selection is still unclear.

The GLCC model excels at determining the most cost-effective openness among provided alternatives. However, it answers the "what" question—what the optimal openness is—rather than the "how" question. The intricacies of how specific parameters influence openness selection remain unclear, hindering a comprehensive understanding of the model's inner workings.

The intricacy of the GLCC model's calculations poses challenges in comprehending the impact of parameters on openness selection. Unlike a pure analytical model, the GLCC model initiates with a network analysis process to establish the refresh schedule for each component and interface. Subsequently, it computes the life-cycle cost based on the generated refresh schedule. Due to the absence of a straightforward analytic solution, understanding the influence of parameters on the life-cycle cost result becomes a complex task.

Based on the demand above, the GLCC predictor is introduced. The idea of the GLCC predictor is by implementing more assumptions, we can eliminate the pre-network analysis process and come up with pure analytical function for the life-cycle cost calculation. Perhaps based on that, we can extract the general conception on the selection of open and closed for the components and interfaces.

Apart from the need for a pure analytic solution, the GLCC predictor addresses practical challenges associated with obtaining precise inputs for the GLCC model. In real-world scenarios, acquiring accurate data for certain parameters, such as development or procurement costs, can be challenging. The GLCC predictor, designed as a pragmatic solution, streamlines input requirements and relies solely on environmental parameters. Its primary goal is to identify the optimal openness among four scenarios, encompassing various combinations of open and closed component/interface configurations.

In the subsequent sections, the fundamental assumptions of the GLCC predictor will be first introduced. Next, the derivation of the GLCC predictors, which is based on the analytic expressions of the GLCC model and the aforementioned assumptions, will be presented. In the last section, the GLCC predictors are applied to the A-RCI case study, comparing the results with those obtained using the GLCC model.

5.1 Assumptions

The assumptions introduced in this section play a crucial role in streamlining the complexity of the GLCC model. This simplification enables the creation of an analytical framework for representing the life-cycle cost of the system. It not only reduces the intricacy but also enhances the clarity of evaluating how various parameters impact the life-cycle cost.

This simplification process hinges on minimizing the variance in the support lives (O_i) of components and interfaces. When there is substantial variability in support lives, it results in disparate refresh dates for individual components and interfaces. As the number of these components and interfaces grows, keeping track of multiple refresh dates becomes progressively challenging. Additionally, formulating a concise and elegant analytical expression for life-cycle cost becomes more complex in such circumstances.

With these assumptions in place, all components and interfaces are systematically

categorized into limited specific groups. Each group shares a same number of refreshes and identical refresh dates. This fundamental categorization significantly streamlines the computation of life-cycle cost within the GLCC model.

Given the refresh review interval is every T years, the assumptions are listed as:

1. Support lives of open components must all be less than T.
2. Support lives of closed components must all be between $[(a\text{-}1)T, aT]$, where a is a positive integer.
3. Support lives of open interfaces must all be between $[(b\text{-}1)T, bT]$, where b is a positive integer.
4. b must be a multiple of a.

In the first assumption, it is impractical to have a refresh cycle shorter than the support lives of components, especially for the purpose of mitigating obsolescence. The overarching goal of assumptions 1 to 3 is to diminish the variability in the support lives of components and interfaces, leading to the synchronization of refresh dates for components or interfaces of the same type. For example, open components are all refreshed every T years in this case. Without external complexities, closed components are refreshed every aT years, and open interfaces are refreshed every bT years. Having the same refresh cycle for components and interfaces of the same type simplifies the process of combining their respective life-cycle costs.

Assumption 4, requiring b to be a multiple of a, assumes critical importance in ensuring that the refresh dates of components align as intended. This alignment proves especially significant when component refreshes are triggered by interfaces. For instance, if a component undergoes refresh every aT years and is also subject to interface-triggered refreshes every bT years, making b a multiple of a ensures that interface-triggered refreshes align harmoniously with the component's original refresh dates. In essence, this coordination simplifies the model by

reducing the variability in refresh intervals, contributing to a more straightforward analytical representation.

In summary, these assumptions ensure that, for every component and interface, the duration of all refresh cycles are the same throughout their respective life cycles. Moreover, components and interfaces within the same category will also have uniform refresh frequencies and identical refresh dates. The forthcoming sections will provide a more comprehensive exploration of how this synchronization not only enhances the model's efficiency but also improves the clarity of cost evaluation and the selection of system openness.

5.2 Development of the GLCC Predictor

This section provides the derivation of the GLCC predictor.

Equation (4.1) and (4.2) have shown that the total life-cycle cost can be represented as the sum of the life-cycle cost of each component and interface, LCC_i. Equation (4.3) also showed that LCC_i is the non-recurring cost, $Cost_{NRE,i}$, plus the sum of the multiplication of discount factors, $f_{k,i}$, and the recurring costs, $Cost_{R,k,i}$.

Given the condition, provided by the assumptions mentioned earlier, the durations of the refresh cycles of component/interface i throughout the life cycle are the same. With the same duration of each refresh cycles ($T_{1,i} = T_{2,i} = \ldots = T_{k,i}$), the recurring cost in each refresh cycle of component/interface i is the same as well. The subscript k can be removed from the recurring costs.

$$Cost_{R,1,i} = Cost_{R,2,i} = Cost_{R,3,i} = \cdots = Cost_{R,k,i} = Cost_{R,i} \tag{5.1}$$

The life-cycle cost of component/interface i can be rewritten as in Equation (5.2), the summation of the discounted recurring cost becomes the recurring cost multiplied by an overall discount factor R_i, which is equal to the summation of all the $f_{k,i}$.

$$LCC_i = Cost_{NRE,i} + \sum_{k=1}^{N_{R,i}} f_{k,i} Cost_{R,k,i} = Cost_{NRE,i} + R_i Cost_{R,i} \tag{5.2}$$

The total life-cycle cost of component/interface *i* becomes,

$$\begin{aligned} LCC_i = {} & C_{Development,i} + C_{Production,i} \\ & + R_i\left(C_{Refresh,i} + C_{Maintenance,i} + C_{BridgeBuy,i} + C_{Hold,i}\right) \end{aligned} \tag{5.3}$$

Replace the individual terms by the results in Table 4.3, Equation (5.3) becomes,

$$\begin{aligned} LCC_i = {} & DC_i + N_{sys}PC_i \\ & + R_i\left\{\left[DC_i + N_{sys}PC_i\right] + \left[N_{sys}\lambda_i PC_i \frac{(1+r)^{T_i} - (1+r)^{T_i - O_i}}{r}\right]\right. \\ & + \left[B * N_{sys}\lambda_i t_i PC_i (1+r)^{t_i}\right] \\ & \left. + \left[N_{sys}\lambda_i C_h PC_i \left(Bt_i \frac{(1+r)^{t_i} - 1}{r} - \frac{(1+r)^{t_i} - rt_i - 1}{r^2}\right)\right]\right\} \end{aligned} \tag{5.4}$$

Rearranging Equation (5.4),

$$\begin{aligned} LCC_i = {} & DC_i + N_{sys}PC_i \\ & + R_i\left\{DC_i\right. \\ & + N_{sys}PC_i\left(1 + \underbrace{\lambda_i \frac{(1+r)^{T_i} - (1+r)^{T_i - O_i}}{r}}_{M_i} + \underbrace{B\lambda_i t_i (1+r)^{t_i}}_{B_i}\right. \\ & \left.\left. + \underbrace{C_h \lambda_i \left(Bt_i \frac{(1+r)^{t_i} - 1}{r} - \frac{(1+r)^{t_i} - rt_i - 1}{r^2}\right)}_{H_i}\right)\right\} \end{aligned} \tag{5.5}$$

By replacing the terms with M_i, B_i and H_i, Equation (5.5) becomes,

$$LCC_i = DC_i + N_{sys}PC_i + R_i\left\{DC_i + N_{sys}PC_i(1 + M_i + B_i + H_i)\right\} \tag{5.6}$$

Where the M_i, B_i and H_i are functions of component failure rate (λ_i), refresh interval (T_i), support life (O_i) and discount rate (r), bridge buy ratio (B),

$$\begin{aligned} M_i &= M(\lambda_i, T_i, O_i, r) \\ B_i &= B(\lambda_i, T_i, O_i, r, B) \\ H_i &= H(\lambda_i, T_i, O_i, r, B, C_h) \end{aligned} \tag{5.7}$$

When there are n components, the total life-cycle cost of these n components can be written as n multiplied by the average life-cycle cost $\overline{LCC}_n$,

$$\sum_{i=1}^{n} LCC_i = n \times \overline{LCC} \tag{5.8}$$

If these n components all have the same refresh interval, e.g., they are within the same category. The average life-cycle cost can be presented as in Equation (5.9). Note that since the n components all share the same refresh dates, they would have the same overall discount factor $R_1 = R_2 = R_3 = \ldots = R_n$. The bar represents the average value, e.g., $\overline{DC}$ is the average development cost of the n components.

$$\overline{LCC} = \overline{DC} + N_{sys}\,\overline{PC} + R_n\left[\overline{DC} + N_{sys}(\overline{PC} + \overline{PC \times M} + \overline{PC \times B} + \overline{PC \times H})\right] \tag{5.9}$$

If these components have similar characteristics, e.g., similar development/procurement cost, failure rate, support life, etc., the average life-cycle cost can be further approximated as in Equation (5.10). Here the average values of each input parameters are used to determine the average life-cycle cost

$$\begin{aligned} \overline{LCC} \cong\ & \overline{DC} + N_{sys}\,\overline{PC} \\ & + R_n\{\overline{DC} \\ & + N_{sys}\left[\overline{PC} + \overline{PC} \times M\left(\bar{\lambda}, T, \bar{O}, r\right) + \overline{PC} \times B\left(\bar{\lambda}, T, \bar{O}, r, B\right)\right. \\ & \left. + \overline{PC} \times H\left(\bar{\lambda}, T, \bar{O}, r, B, C_h\right)\right]\} \end{aligned} \tag{5.10}$$

For interfaces, the procurement cost is equal to zero, the average life-cycle cost becomes,

$$\overline{LCC} \cong \overline{DC} + R_n\overline{DC} \tag{5.11}$$

Equations (5.10) and (5.11) outline the method for calculating the average life-cycle cost of components or interfaces within the same category. This enables us to express the total life-cycle cost as the product of the quantity of each group of components/interfaces and their respective average life-cycle costs. The question is, how many groups are there for the components and the interfaces?

In the GLCC predictor, it is assumed that open components, closed components, open interfaces, and closed interfaces share similar characteristics. Building upon this assumption, along with the support lives assumption discussed earlier, we can effectively categorize all components into just three groups, and similarly, interfaces can be divided into three distinct groups.

In the GLCC predictor, components fall into three groups: open components, affected closed components, and non-affected closed components. Open components follow a *T*-year refresh cycle. The second group, affected closed components, consists of closed components connected to open components through closed interfaces. When open components are refreshed, affected closed components also undergo refreshes, every *T* years. The last group, non-affected closed components, encompasses closed components whose refresh dates remain independent of open components. These non-affected closed components may not have connections with open components, or if they do, they connect through open interfaces. Their refresh dates are solely triggered by the obsolescence of the closed components. In summary, there are two distinct refresh date patterns: one shared by open components and affected closed components, and the other exclusive to non-affected components.

Moreover, interfaces can also be categorized into three groups: open interfaces, open-components-affected closed interfaces, and closed-components-affected closed interfaces. Open interfaces adhere to a uniform refresh cycle based on their respective support lives. In contrast, closed interfaces are considered to have zero support lives, and their refresh dates are contingent on the support lives of the connected components. If at least one of the connected components is open, the closed interface is refreshed concurrently with the open components. However, if both connected components are closed, the refreshes of the closed interface are triggered by the closed component's obsolescence. To summarize, open interfaces have refresh dates based on open interface support lives, open-components-affected closed interfaces rely on open component support lives for their refresh dates, and closed-components-affected closed interfaces' refresh dates depend on the support lives of the connected closed components.

Thus, the life-cycle cost can be expressed as the sum of the multiplication of the average life-cycle cost and the number of components or interfaces of each group. The total life-cycle cost is presented in Equation (5.12), where n_1, n_{21}, n_{22}, n_3, n_{41} and n_{42} represent the number of the open components, the affected closed components, the non-affected closed components, the open interfaces, the open-components-affected closed interfaces and the closed-components-affected closed interfaces.

$$LCC = n_1\overline{LCC}_1 + n_{21}\overline{LCC}_{21} + n_{22}\overline{LCC}_{22} + n_3\overline{LCC}_3 + n_{41}\overline{LCC}_{41} + n_{42}\overline{LCC}_{42} \tag{5.12}$$

Table 5.1 presents the details of Equation (5.12). Note that since the open components and the affected closed components have the same refresh cycles, their overall discounted factors are the same, R_1. The overall discount factor of the non-affected closed component, R_2, is solely based on the support life of the closed components. Moreover, the refreshes of the open-components-affected closed interfaces are triggered by the refresh of the open components, so the corresponding overall discounted factor is the same as the open components', R_1. On the other hand, the refreshes of the closed-components-affected closed interfaces are triggered by

the refresh of the closed components, so the corresponding overall discounted factor is the same as the closed components', R_2. The refresh of the open interface is triggered by the obsolescence of the open interface. The overall discount factor is shown independently as R_3.

Table 5.1 Details of Equation (5.12)

Category names	**Number of components/ interfaces**	**Average life-cycle cost $\overline{LCC}$**
Open components	n_1	$(DC_1 + N_{sys}PC_1) + R_1\left(DC_1 + N_{sys}PC_1(1 + M_1 + B_1 + H_1)\right)$
Affected closed components	n_{21}	$(DC_2 + N_{sys}PC_2) + R_1\left(DC_2 + N_{sys}PC_2(1 + M_2^* + B_2^* + H_2^*)\right)$
Non-affected closed components	n_{22}	$(DC_2 + N_{sys}PC_2) + R_2\left(DC_2 + N_{sys}PC_2(1 + M_2 + B_2 + H_2)\right)$
Open interfaces	n_3	$DC_3 + R_3DC_3$
Open-components-affected closed interfaces	n_{41}	$DC_4 + R_1DC_4$
Closed-components-affected closed interfaces	n_{42}	$DC_4 + R_2DC_4$

Upon formulating the life-cycle cost in an analytical framework, the quest for optimal openness, aiming at minimizing the life-cycle cost, transforms into an optimization problem as presented in Equation (5.13). This, in essence, evolves into a mixed integer linear programming problem. The variables under consideration are n_1, n_{21}, n_{22}, n_3, n_{41} and n_{42}, while the objective function is the life-cycle cost depicted in Equation (5.13). The coefficients $\overline{LCC}_1, \overline{LCC}_{21}, \overline{LCC}_{22}, \overline{LCC}_3, \overline{LCC}_{41}$ and $\overline{LCC}_{42}$ are determined by environmental parameters, with p representing the total number of components in the system and q representing the total number of interfaces. The variables n_i denote the quantity of different types of components or interfaces and, therefore, must be non-negative integers.

Beyond the mathematical constraints explicitly outlined in Equation (5.13), additional constraints arise from the system architecture network. The inherent nature of the architecture network imposes restrictions on the relationships between variables. Certain combinations of variables may be infeasible due to the network's topological characteristics. For instance, if

there are no open components in the architecture, the existence of affected closed components is precluded. In other words, a solution like [0, p, 0] for [n_1, n_{21}, n_{22}] does not exist. These network constraints, although intricate and challenging to articulate analytically, play a pivotal role.

$$\min \overline{LCC}_1 n_1 + \overline{LCC}_{21} n_{21} + \overline{LCC}_{22} n_{22} + \overline{LCC}_3 n_3 + \overline{LCC}_{41} n_{41} + \overline{LCC}_{42} n_{42}$$

$$\begin{gathered} subject\ to\ n_1 + n_{21} + n_{22} = p \\ n_3 + n_{41} + n_{42} = q \\ n_1, n_{21}, n_{22}, n_3, n_{41}, n_{42} \in \mathbb{N}_0 \\ \text{Other network constraints} \\ p\text{: total number of components} \\ q\text{: total number of interfaces} \end{gathered} \tag{5.13}$$

Given the inherent network constraints, solving the integer linear programming problem directly is challenging. Nevertheless, the coefficients offer a directional guide for determining optimal openness. In essence, this integer linear programming problem involves assigning p components and q interfaces as open or closed to minimize the life-cycle cost. Common intuition suggests that components should be assigned based on the least coefficient, specifically the average life-cycle cost.

For components, the relative magnitudes of the coefficients $\overline{LCC}_1$, $\overline{LCC}_{21}$, and $\overline{LCC}_{22}$ depend on open/closed components' development/procurement costs and support lives. If the average life-cycle cost of open components, $\overline{LCC}_1$, is the least, opting for an all-open configuration is sensible. Similarly, when the non-affected closed component, $\overline{LCC}_{22}$, has the lowest cost, an all-closed configuration is viable. However, for the affected closed components' average life-cycle cost, $\overline{LCC}_{21}$, complexities arise. Since affected closed components are linked

to open components via closed interfaces, their existence depends on the presence of open components and closed interfaces in the system architecture. If affected closed components have the least average life-cycle cost, it doesn't necessarily mean making all components affected closed is the optimal solution. This suggests a possibility of employing a hybrid combination of open and closed components in the system architecture.

Concerning the interface coefficients ($\overline{LCC}_3$, $\overline{LCC}_{41}$, and $\overline{LCC}_{42}$), $\overline{LCC}_{41}$ consistently exceeds $\overline{LCC}_{42}$ due to the more frequent refresh of the closed interface. When solely focusing on the values of the interfaces' average life-cycle cost, the decision to assign interfaces as open or closed should be based on the relative magnitudes of $\overline{LCC}_3$ and $\overline{LCC}_{42}$. However, it's essential to recognize that the openness of the components can impact the cost of the interfaces. In the presence of open components, the average life-cycle cost of closed interfaces becomes $\overline{LCC}_{41}$, while with closed components, it becomes $\overline{LCC}_{42}$. Therefore, the influence of component openness must be factored in when determining the openness of the interfaces.

Moreover, the comparative significance of the average life-cycle cost between components and interfaces plays a crucial role in determining openness allocation. This aspect dictates whether components or interfaces exert a more dominant influence on the overall life-cycle cost. For instance, if components have a relatively larger scale, meaning $\overline{LCC}_1, \overline{LCC}_{21}$, and $\overline{LCC}_{22}$ are considerably greater than $\overline{LCC}_3, \overline{LCC}_{41}$, and $\overline{LCC}_{42}$, priority in openness selection should be assigned to components. The openness selection of interfaces is intended to complement the chosen openness of components. In light of the foregoing discussion, we propose the architecture openness selection process, considering the interdependency between components and interfaces.

In the process of selecting openness, there are five potential solutions: [open components, open interfaces], [open components, closed interfaces], [closed components, open interfaces],

[closed components, closed interfaces], and a hybrid solution. The first four solutions involve fully open or fully closed components and interfaces, while the fifth solution suggests a hybrid approach with partial openness in components and interfaces. This process comprises two steps. Initially, the potential for a hybrid solution is identified by comparing the values of $\overline{LCC}_1$, $\overline{LCC}_{21}$, and $\overline{LCC}_{22}$. If $\overline{LCC}_{21}$ is the minimum, it indicates the possibility, though not absolute, that a hybrid openness is optimal. A more in-depth analysis using the GLCC model is then conducted to confirm the hybrid solution and determine the optimal degree of openness. If the values of $\overline{LCC}_1$ or $\overline{LCC}_{22}$ are the minimum, it suggests that the solution lies among the four fully open/closed alternatives. Consequently, the process advances to step two, where the life-cycle costs of these four alternatives are calculated based on Equation (5.12). This calculation incorporates the average life-cycle cost $\overline{(LCC)}$ and the quantities of components and interfaces denoted as p and q. After computing the life-cycle costs for these four scenarios, the most cost-effective solution (openness configuration) is determined by selecting the one with the lowest cost.

The openness selection process is summarized in Figure 5.1. These four life-cycle cost values serve as the GLCC predictors, aiding in the determination of preferences between open and closed components and interfaces. This approach facilitates a swift and efficient decision-making process, proving valuable for practical applications in system design and optimization. The underlying concept of the GLCC predictor involves using only environmental parameters, bypassing intricate considerations in system architecture, to predict the cost-effectiveness of open or closed components and interfaces. It operates as a straightforward plug-and-play operation, allowing for prompt execution.

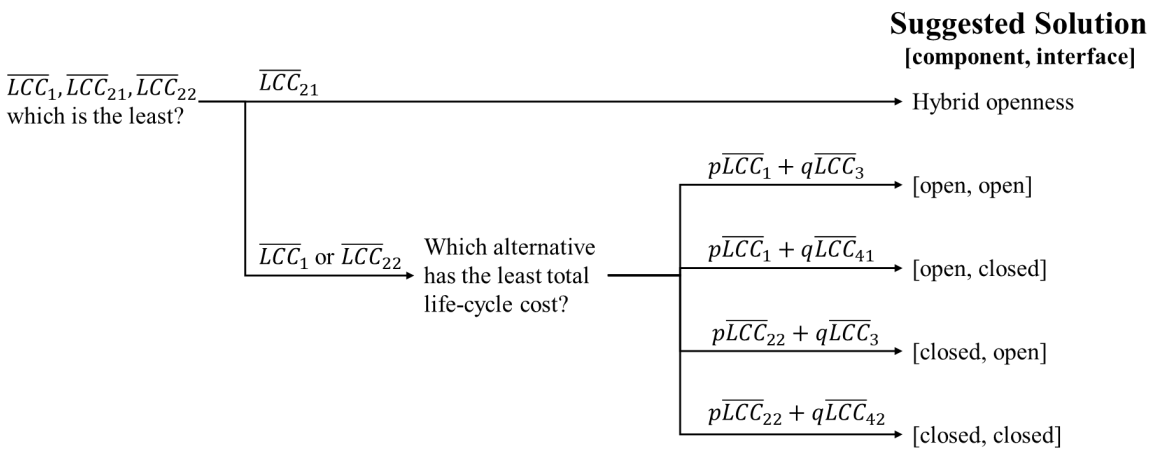


Figure 5.1 The openness selection process based on the GLCC predictors.

In essence, the GLCC predictor estimates the life-cycle cost by considering the average life-cycle cost of open and closed components and interfaces. In contrast to the meticulous process in the GLCC model, which examines each component and interface, determining their refresh intervals and calculating individual life-cycle costs, the GLCC predictor streamlines the procedure by averaging the parameters of components and interfaces to derive the average life-cycle cost. While this approach has its limitations, such as the prerequisite for support lives of the same component category to align within a similar range of refresh cycles and potential errors during parameter averaging and subsequent life-cycle cost calculation, it presents a time-saving advantage by sidestepping the need for detailed scrutiny of every component in the system architecture. Subsequent sections will offer a more in-depth comparative analysis of the outcomes generated by these two approaches.

5.3 Discussion on the Analytic Formulation of the GLCC Predictor

In this section, the focus is on examining the mathematical forms of the GLCC predictor. As discussed in the preceding section, the process of the GLCC predictor revolves around assessing the relative magnitudes of the six generated average life-cycle cost values (from $\overline{LCC}_1$ to $\overline{LCC}_{42}$). Taking an analytical perspective, the formulations of the average life-cycle cost will be elaborated upon, exploring how input parameters influence these values. This

exploration aims to provide insights into the factors determining the selection of the system's openness.

The examination begins by comparing the first three average life-cycle costs associated with open/closed components, marking the initial step in the GLCC predictor's openness selection process. Figure 5.2 visually represents this comparison. In this illustrative case, 1000 samples are generated with varied development and procurement costs of the closed component, while keeping other input parameters constant. The horizontal axis denotes the development cost ratio between the closed and open components ($\frac{DC_{closed}}{DC_{open}}$), and the vertical axis represents the procurement cost ratio of the closed and open components ($\frac{PC_{closed}}{PC_{open}}$). Each point on the graph signifies a unique scenario with a specific combination of closed component development and procurement costs.

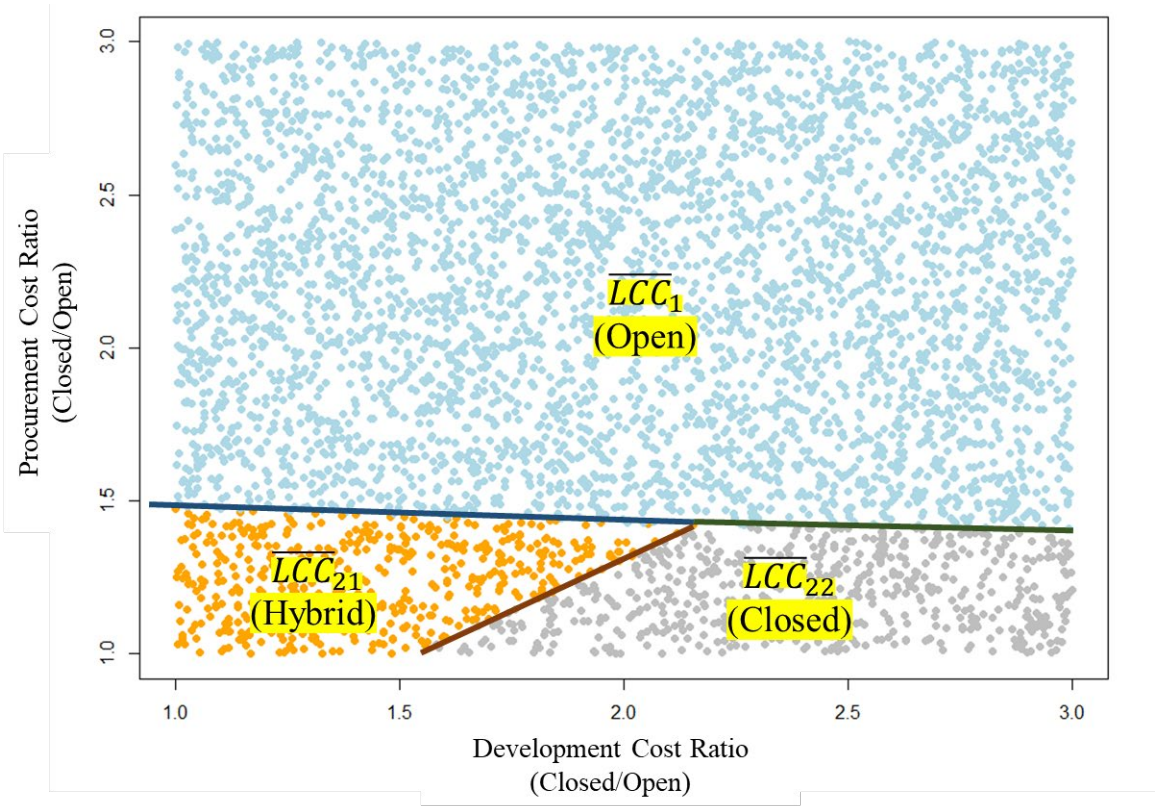


Figure 5.2 Visualization of the comparison of $\overline{LCC_1}$, $\overline{LCC_{21}}$, and $\overline{LCC_{22}}$.

Figure 5.2 shows that with a given combination of closed component development and procurement cost, what component average life-cycle cost, i.e., $\overline{LCC}_1$, $\overline{LCC}_{21}$, and $\overline{LCC}_{22}$, would be the minimum, open or closed component. The color of the point indicates which average life-cycle cost is the least. If a combination sample of the development and procurement cost lies in the light blue area, it suggests that using open components is more cost-effective. If the sample is in the gray area, using closed components is less expensive. If the sample is in the orange area, there is a possibility that the system is more cost-effective in a hybrid openness.

From this specific example, we can generally conclude that if the procurement cost of the closed component is 1.5 times greater than the open component, it is better to use open components. However, do note that this figure only considered the average life-cycle cost of the components and does not consider the influence of interfaces. It does not guarantee the component openness selection since the component and interface should be considered jointly.

On the other hand, it can also be observed that the regions in Figure 5.2 is divided by three boundary lines: $\overline{LCC}_1 = \overline{LCC}_{21}$, $\overline{LCC}_{21} = \overline{LCC}_{22}$, $\overline{LCC}_1 = \overline{LCC}_{22}$. On these lines, the values of the average life-cycle cost are the same. These lines determine the size of the areas, how expensive the closed components can be so that implementing closed components can still be more cost-effective.

To examine how the input parameters affect these lines, the analytic forms of the lines have to be determined. Equation (5.14) first rearranges the average life-cycle costs, i.e., $\overline{LCC}_1$, $\overline{LCC}_{21}$, and $\overline{LCC}_{22}$. The common terms of the development cost and procurement cost are extracted. Besides, for equation simplicity, the term (1+M+B+H) is replaced by another symbol Q.

$$\begin{cases} \overline{LCC}_1 = DC_1(1+R_1) + N_{sys}PC_1\left(1 + R_1\underbrace{(1+M_1+B_1+H_1)}_{Q_1}\right) \\ \overline{LCC}_{21} = DC_2(1+R_1) + N_{sys}PC_2\left(1 + R_1\underbrace{(1+M_2^*+B_2^*+H_2^*)}_{Q_2^*}\right) \\ \overline{LCC}_{22} = DC_2(1+R_2) + N_{sys}PC_2\left(1 + R_2\underbrace{(1+M_2+B_2+H_2)}_{Q_2}\right) \end{cases} \tag{5.14}$$

Let $DC_2=\alpha_1 DC_1$, $PC_2=\alpha_2 PC_1$. α_1 and α_2 are the cost factor between the closed and the open components, indicating how much more expensive the closed components are in the development and procurement costs. They are also the axis variables in Figure 5.2. By letting the three functions be equal with one another, we can get the equation of the three boundary lines. Here, α_1 and α_2 are the only variables, and other parameters are fixed, including the development and procurement cost of the open components. Therefore, the average life-cycle cost of the open components, $\overline{LCC}_1$, is also a constant.

$$\begin{aligned} \overline{LCC}_1 = \overline{LCC}_{21}, \overline{LCC}_1 &= DC_1(1+R_1)\alpha_1 + N_{sys}PC_1(1+R_1Q_2^*)\alpha_2 \\ \overline{LCC}_1 = \overline{LCC}_{22}, \overline{LCC}_1 &= DC_1(1+R_2)\alpha_1 + N_{sys}PC_1(1+R_2Q_2)\alpha_2 \\ \overline{LCC}_{21} = \overline{LCC}_{22}, 0 &= DC_1(R_1-R_2)\alpha_1 + N_{sys}PC_1(R_1Q_2^* - R_2Q_2)\alpha_2 \end{aligned} \tag{5.15}$$

In the initial two boundary line equations, the coefficients of α_1 and α_2 are consistently positive, given the inherent positivity of DC, R, N_{sys}, PC and Q values. The constant $\overline{LCC}_1$ is also positive, signifying that these lines traverse the first quadrant and possess negative slopes. These lines serve as boundaries that demarcate open and closed components. The region enclosed by these boundary lines, along with α_1=1 and α_2=1, signifies the domain where closed components exhibit a more economical average life-cycle cost. This aligns with our intuitive understanding that the space for closed components is constrained. The third boundary line segregates the area between affected and non-affected closed components, appearing as a line passing through the origin with a slope that can be either positive or negative. When the slope is negative, the closed component domain becomes all non-affected closed component and

there is no possibility for the hybrid openness to be the more cost-effective alternative.

The slopes of the boundary lines differentiating open and closed components are determined by the expressions $-\frac{DC_1(R_1+1)}{N_{sys}PC_1(1+R_1Q_2^*)}$ and $-\frac{DC_1(R_2+1)}{N_{sys}PC_1(1+R_2Q_2)}$. This suggests that when the development cost is relatively smaller than the procurement cost (without specifying the absolute values), the slopes of these lines become flatter. This characteristic is evident in Figure 5.2, where flatter boundary lines result in a rectangular-like shape for the closed component area. The impact of the development cost ratio is less pronounced compared to the procurement cost ratio, aligning with the observation that the development cost is relatively smaller than the procurement cost.

The third boundary line delineates the region between affected-closed components and non-affected components, also serving as the boundary indicating the possibility of hybrid openness. If the slope of the boundary line is positive, as depicted in Figure 5.2, there is a potential for hybrid openness to be optimal. The slope can be expressed as $-\frac{DC_1(R_1-R_2)}{N_{sys}PC_1(R_1Q_2^*-R_2Q_2)}$. The numerator is consistently positive since R_1 is always greater than R_2. If $R_1\ Q_2^* - R_2\ Q_2$ is also positive, the slope becomes negative, signifying that $\overline{(LCC)}_{22}$ is consistently greater than $\overline{(LCC)}_{21}$, and there is no possibility of employing a hybrid openness.

5.4 The A-RCI Case Demonstration

In this section, we undertake a comparison of the life-cycle results obtained through both the GLCC model and the GLCC predictors to discern any discrepancies. Utilizing the same A-RCI case employed in the GLCC model, the subsequent analysis will delve into the factors contributing to the differences observed in the outcomes of these two approaches.

The inputs for the GLCC predictor are documented in Table 5.2 and were initially employed in the GLCC model's first example, as illustrated in Figure 4.12. Instead of featuring the

specific development cost, procurement cost, and support lives for individual components and interfaces, the GLCC predictor relies on the averages of these costs and support lives. Following the framework presented in Figure 5.1, the relative life-cycle cost of the four scenarios ([open components, open interfaces], [open components, closed interfaces], [closed components, open interfaces], [closed components, closed interfaces]) can be computed.

Table 5.2 The Inputs for the GLCC Predictor

	Inputs	Value	Description
General parameters	EOS	50	End-of-support year
	N_{sys}	50	The number of the supported systems
	λ	0.00370	The average failure rate of the component
	r	0.07	Discount rate
	T	5	The predetermined refresh interval
	B	1.2	The bridge buy ratio
	C_h	0.2	The unit holding cost ratio per component per year
Component parameters	DC_{open}	$7.07M	Average development cost of open component
	PC_{open}	$1.63M	Average procurement cost of open component
	O_{open}	3	Average support life of open component
	DC_{closed}	$46.48M	Average development cost of closed component
	PC_{closed}	$8.01M	Average procurement cost of closed component
	O_{closed}	6.74	Average support life of closed component
Interface parameters	DC_{open}	$220K	Average development cost of open interface
	O_{open}	15	Average support life of open interface
	DC_{closed}	$81.67K	Average development cost of closed interface
	O_{closed}	0.001	Average support life of closed interface

Figures 5.3 present a comparative analysis between the outcomes of the GLCC model and the GLCC predictor. The horizontal axis delineates four distinct scenarios: [open components, open interfaces], [open components, closed interfaces], [closed components, open interfaces], and [closed components, closed interfaces]. The vertical axis represents the estimated life-cycle cost, with the blue (left) bars signifying results from the GLCC model and the green (right) bars representing those from the GLCC predictor. The numerical values atop each bar denote the mismatch rate[13] calculated as the difference between the life-cycle cost and the GLCC model's life-cycle cost, divided by the latter.

[13] The goal here is to understand how the assumptions of the GLCC predictor can introduce error to the GLCC model. The mismatch rate here uses the GLCC model result as the baseline and is based on the difference between the two approaches. It has nothing to do with the actual life-cycle cost.

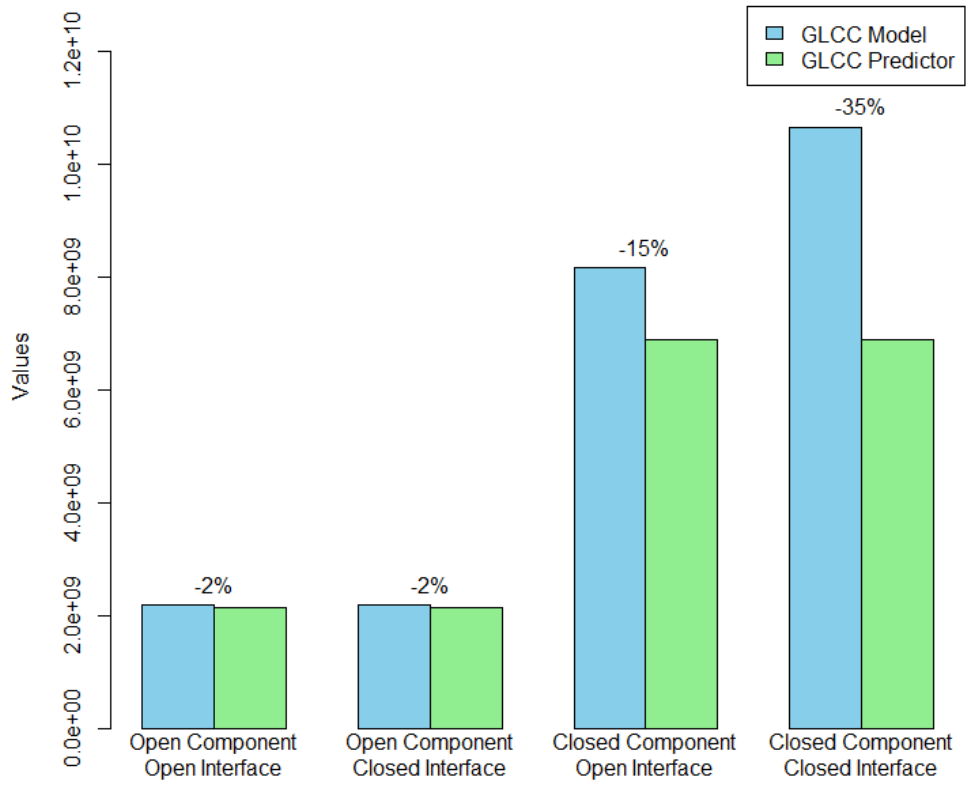


Figure 5.3 The (relative) life-cycle cost comparison of the A-RCI case between the GLCC model and the GLCC predictor.

According to the GLCC predictor results (the right green bars) in Figure 5.3, the first two categories and the last two categories exhibit similar life-cycle cost values, suggesting that changes in interface openness have a relatively minor impact on the life-cycle cost. Conversely, alterations in component openness wield more influence, resulting in approximately a $4 billion life-cycle cost difference. This discrepancy can be attributed to the significant scale difference between the development costs of components and interfaces, with interface development costs ranging from $80K to $200K, while component development/procurement costs can be $10 to $50 million.

For the GLCC model (the left blue bars), the life-cycle cost difference is comparable in the first and second categories but significantly diverges between the third and fourth categories,

amounting to around a $3 billion difference. This underscores the substantial role that interface openness plays in the life-cycle cost when all components are closed.

Combining the results of both approaches reveals that the GLCC predictor provides a reasonable estimate for the first two categories, with mismatch rates around 2%. However, it struggles in the third and fourth categories, exhibiting mismatch rates of 15% and 35%, respectively. Furthermore, the GLCC predictor fails to capture the substantial differences between the third and fourth categories. These comparative results highlight that directly employing inputs from the GLCC model without considering the GLCC predictor limitation/assumptions can introduce mismatches into life-cycle cost estimations.

The discrepancies observed in the comparison results between the GLCC model and the GLCC predictor can be attributed to two key assumptions made during the development of the predictor. The first assumption posits that the support lives of open or closed components fall within the same refresh interval, ensuring that all components within each category share identical refresh intervals. Unlike the GLCC model, which calculates the life-cycle cost for individual components and interfaces and subsequently sums them up (as seen in Equations (5.10) to (5.11)), the GLCC predictor determines the average life-cycle cost using the average values of the parameters. However, if there is significant variance in these parameters, the average life-cycle cost may deviate.

Upon closer examination of the inputs for closed components, it becomes evident that one closed component has a support life of 4.37 years, while the remaining closed components have support lives ranging from 5 to 10 years. This discrepancy contradicts the first assumption. In scenarios where the architecture adopts all closed components, the GLCC model refreshes the components separately, with the one having a 4.37-year support life refreshed every 5 years, while the others are refreshed every 10 years. In contrast, the GLCC predictor averages the

support lives of closed components to be 6.74 years, placing them in the 5 to 10-year range. Consequently, all closed components are refreshed every 10 years in the predictor. With the one component refreshed every 5 years, the GLCC model incurs a higher total life-cycle cost.

In the fourth scenario, where all interfaces are closed, and all components are refreshed together, the GLCC model refreshes all closed components together every 5 years. Conversely, the GLCC predictor assumes that all components have a 6.74-year support life and are refreshed every 10 years, resulting in a significantly lower life-cycle cost.

In summary, the GLCC predictor fails to account for the variance in support lives, specifically overlooking the presence of one closed component with a 4.37-year support life and a more frequent refresh rate. Instead, it incorrectly assumes that all components have a 6.74-year support life, leading to the underestimation of the life-cycle costs for closed components.

To confirm the validity of our conclusion, we deliberately adjusted the input dataset by modifying the 4.37-year support life to 6.37 years, ensuring that the support lives of the closed components now fall within the range of 5 to 10 years. Figure 5.4 depicts the comparison results based on the updated inputs, revealing a notable reduction in mismatch rates for the closed components scenarios—from 15% and 35% to 7%.

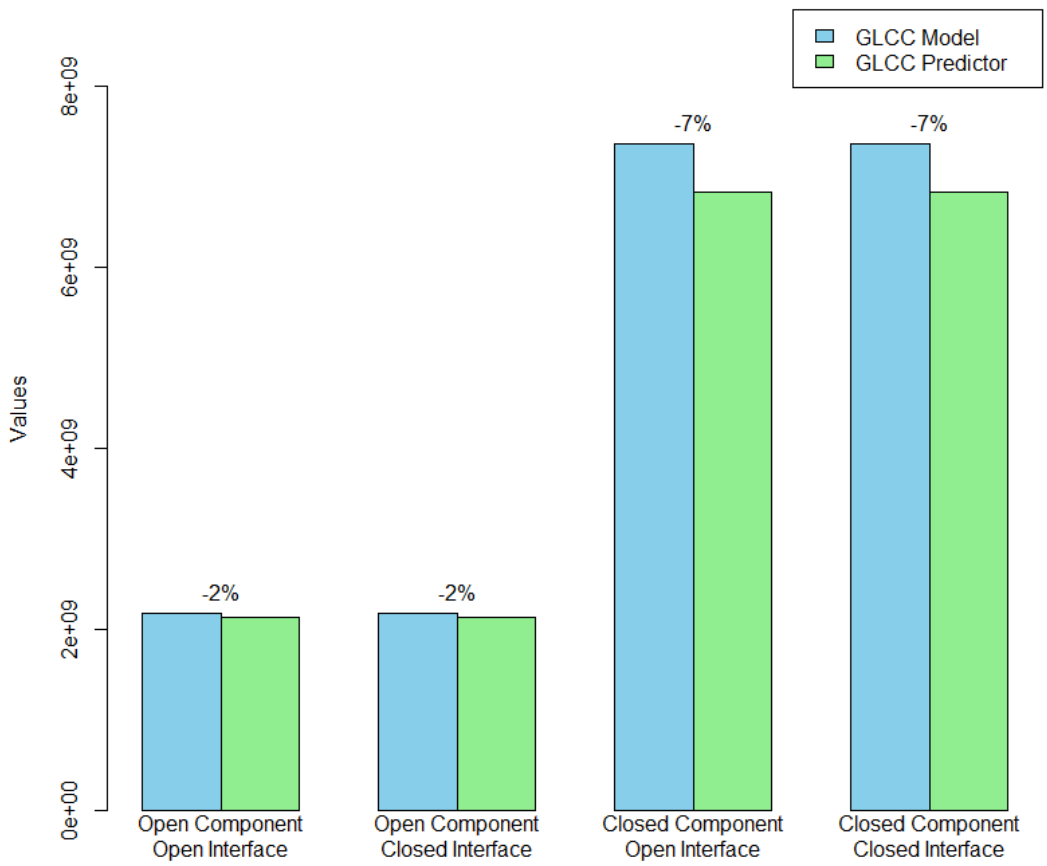


Figure 5.4 The (relative) life-cycle cost comparison of the A-RCI case between the GLCC model and the GLCC predictor—with updated support life.

Despite the readjustment of support lives in the GLCC predictor to align with assumptions, there still persists a mismatch rate of approximately 7% in the closed component scenarios. As previously discussed, these mismatches stem from variations in the inputs. In Equations (5.10) to (5.11), we approximated the average of the product of *PC* and *M* by $\overline{PC}M(\bar{\lambda}, T, \bar{O}, r)$, the average of the product of *PC* and *B* by $\overline{PC}B(\bar{\lambda}, T, \bar{O}, r, B)$ and the average of the product of *PC* and *H* by $\overline{PC}H(\bar{\lambda}, T, \bar{O}, r, B, C_h)$. Since T, r, B and C_h are fixed values, it can be concluded that the variances in procurement costs (*PC*), failure rates (λ), and support lives (*O*) are the factors contributing to the introduced mismatches. It is important to note that, given that the development cost (*DC*) does not involve any multiplication computation, the variance in development cost does not introduce mismatches between the GLCC model and the GLCC

predictors.

Table 5.3 presents the coefficient of variance (*CV*) for crucial inputs such as procurement costs (*PC*), failure rates (λ), and support lives (*O*), calculated from the data in Figure 4.12. The coefficient of variance (*CV*) quantifies the ratio of standard deviation to the mean, offering a measure of data variability. Analysis of the *CV* table reveals that, among the three parameters concerning open/closed components, closed components exhibit an overall higher variance compared to open components and interfaces. This finding provides a plausible explanation for the larger mismatch rates observed in scenarios involving closed components.

Table 5.3 The Coefficient of Variance (CV) of Procurement Costs (PC), Failure Rates (λ), and Support Lives (O).

	Support lives (*O*)	Procurement costs (*PC*)	Failure rates (λ)
Open component	0	0.4983	0.5897
Closed component	0.2355	0.9153	0.5897

To examine the impact of parameter variance on the introduced mismatch, a supplementary numerical experiment was conducted using the A-RCI case as a basis. A total of 2000 samples were generated, wherein parameters—procurement costs (*PC*), failure rates (λ), and support lives (*O*)—were systematically varied while keeping the remaining parameters constant. To maintain alignment with assumptions, the support lives were confined within the same refresh interval. The coefficient of variance (*CV*) for each parameter in every sample was computed and aggregated. Subsequently, these randomly generated parameters were input into both the GLCC predictor and the GLCC model, producing life-cycle costs for the four scenarios (as in Figures 5.3 and 5.4). Mismatch rates for each scenario in each sample were computed based on the corresponding life-cycle cost results.

For every scenario in each sample, the means of the *CV*s of the inputs were computed, and the mismatch rate between the GLCC model and the GLCC predictor was calculated. The

aggregated *CV*s and mismatch rates were plotted in Figure 5.5, illustrating the relationship between parameter variance and mismatch rates. As the *CV*s increased, the variance in mismatch rates expanded. The distribution of data points in Figure 5.5 illustrated how much the GLCC predictor could deviate from the GLCC model based on the given *CV*s of the parameters. Additionally, based on the data points in Figure 5.5, an empirical 95% boundary was estimated. This boundary encapsulates 95% of the sample points, implying that 95% of the results generated by the GLCC predictor should fall within the boundary. The boundary serves as the mismatch range of the results generated by the GLCC predictor.

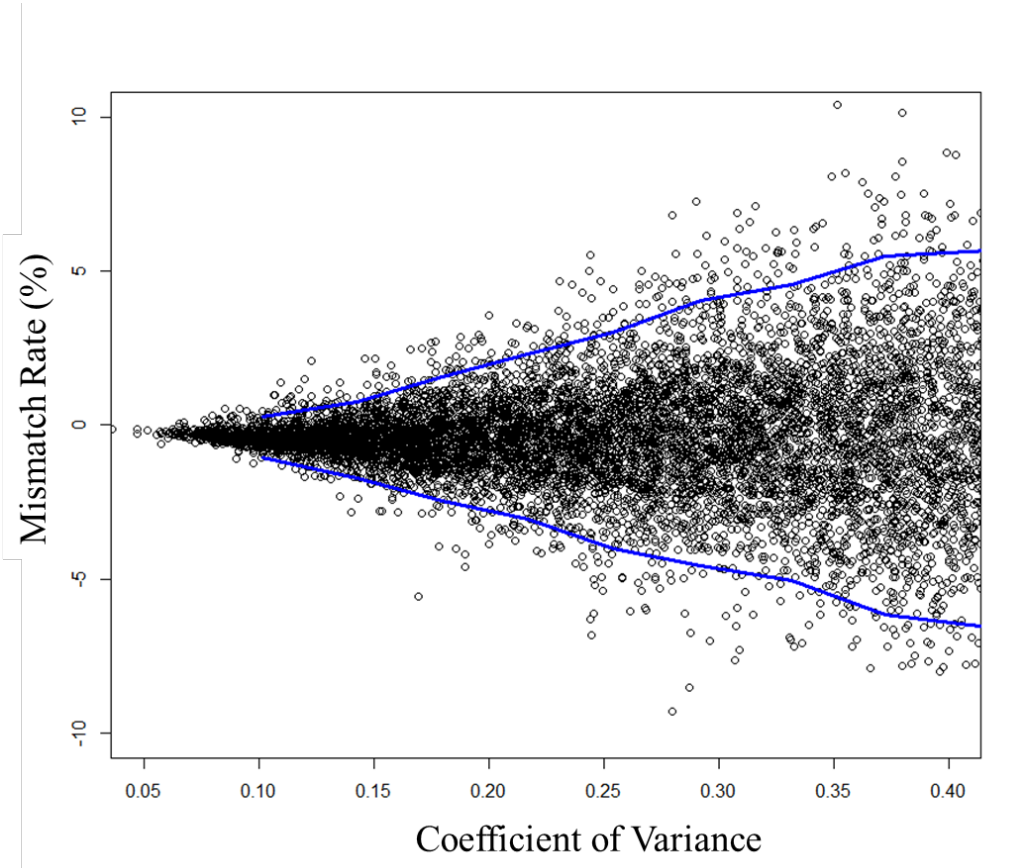


Figure 5.5 The mismatch rate of the GLCC predictor over the coefficients of variance.

Figure 5.6 depicted the relationship between parameter variance (*CV*) and the mismatch range. The mismatch range is calculated based on the interval between the upper bound and lower bound of the 95% boundary in Figure 5.5. This relationship is practical for determining the significance of differences between two alternatives generated by the GLCC predictor.

When the GLCC predictor provides only point estimators, discerning if one alternative is less expensive than the other is challenging, as the difference may lie within the mismatch range. However, with the inherent relationship between *CV* and mismatch rate, using the GLCC predictor equips each point estimate with a mismatch range, offering a more comprehensive predicted result. When comparing two alternatives based on their point estimators and mismatch ranges, conclusions can be drawn regarding the significance of the difference. Overlapping mismatch ranges suggest no significant difference, while non-overlapping ranges indicate that one is significantly less expensive than the other.

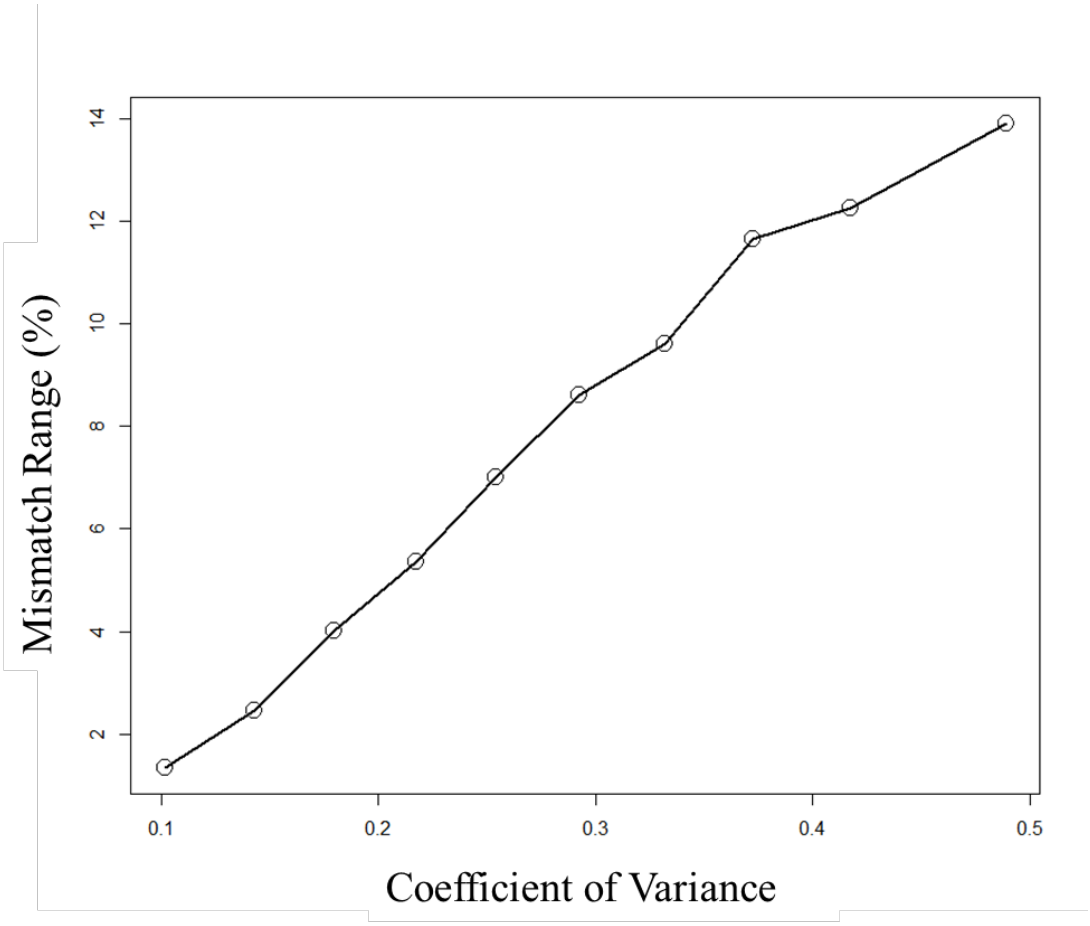


Figure 5.6 The mismatch range (95% interval) of the GLCC predictor over the coefficients of variance.

The comparison of the mismatch rate between the GLCC model and predictor in Figure 5.6 offers a tentative guide for decision-making in the specific A-RCI case when implementing the GLCC predictor approach. However, the empirical 95% boundary, derived from the specific

A-RCI case, is grounded in empirical observations and may lack generalizability across diverse cases. The mismatch rate depicted in Figure 5.6 may hold validity for the A-RCI case but could become less applicable in scenarios with different architectures. While the GLCC predictor formulation highlights parameters whose variance might lead to mismatches, the precise relationship between these variances and the mismatch boundary remains unclear. The current practice involves taking the mean of the coefficient of variations (*CV*s) and assuming a relationship between *CV* mean and the mismatch boundary. While practical for specific cases, its generalizability raises questions. Future endeavors should explore statistical methodologies to establish a more nuanced model between variance and the mismatch boundary. Systematically incorporating a broader range of cases in this exploration can yield a more robust and universally applicable boundary, enhancing its reliability in assessing life-cycle costs using the GLCC predictor across various contexts.

5.5 Conclusion

The GLCC predictor extends the capabilities of the GLCC model by incorporating additional assumptions related to refresh intervals and parameter variances. This enhancement enables the expression of average life-cycle costs for open/closed components/interfaces in analytic forms, allowing for a more comprehensive analysis of parameter influences on openness selection. Despite the advantages of simplicity and speed associated with the GLCC predictor, there exists a trade-off in precision, rendering it particularly suitable for scenarios where a quicker, less precise estimation is deemed acceptable.

The A-RCI case study underscores the GLCC predictor's value in estimating life-cycle costs and guiding system openness decisions. By streamlining the intricate process of individual component and interface assessment, it provides a swift and efficient means of decision-making. Moreover, the analytic formulation of the GLCC predictor offers insights into how parameters

influence life-cycle costs. However, certain assumptions, particularly those pertaining to the range of support life (O) and parameter variance, especially in procurement cost (PC), support life (O), and failure rate (λ), introduce limitations that can impact prediction accuracy. The comparison with the GLCC model illuminates potential mismatches stemming from these assumptions, posing challenges for decision-making when the accuracy range of the estimator is unknown. Consequently, we propose an approach associated with the empirical 95% boundary, offering a practical framework for understanding the significance of differences between alternatives. As a predictive model, it aids in assessing the relative impact of openness configurations on life-cycle costs, facilitating informed decision-making in system design and optimization.

In summary, the GLCC predictor presents promising alternatives to the simulation methods and the GLCC model, providing analytical solution and faster computation speeds to estimate life-cycle costs. The choice between these approaches hinges on the specific requirements (speed, precision and input details) of the decision-making process, striking a balance between the need for precision and the demand for efficiency in system design and optimization.

Chapter 6 Summary, Contributions and Future Work

6.1 Summary

The goals of implementing an open system architecture (OSA) are often defined in qualitative terms. Often, it is taken for granted that the use of open system architecture (OSA) decreases the total life-cycle cost of a system. However, there is a lack of studies quantifying the cost avoidance and assessing the circumstances under which this assumption is true.

This dissertation introduces a framework for quantitatively analyzing Open System Architecture (OSA) by modeling the life-cycle cost difference associated with system openness. Openness is defined as the proportion of Commercial Off-The-Shelf (COTS) components and open standards utilized among all components and interfaces. Three distinct approaches are developed within this dissertation: a detailed simulation, the GLCC model, and the GLCC predictor. The detailed simulation approach involves generating discrete events—development, production, refresh, obsolescence, and maintenance events—over the life cycle, with cumulative events contributing to the overall life-cycle cost. The GLCC model, a hybrid approach, utilizes network analysis to assess refresh events and calculates costs using analytical functions. The GLCC predictor is a purely analytical model capable of calculating relative life-cycle costs but is restricted to four specific openness scenarios, imposing the most stringent constraints and assumptions.

Table 6.1 Comparison of Solution Approaches

Comparing Items	Simulation	GLCC Model	GLCC Predictor
Principle/Method	Discrete-event simulation	Hybrid (Network analysis + Analytic functions)	Analytic formulation
Computation Speed	Relatively slow, especially for complex systems.	Faster than simulation but may still require some computational resources.	Rapid computation, providing quick insights.

Complexity	High, involves detailed event generation.	Moderate, combines network analysis and analytic functions.	Low, relies on simplified assumptions.
Parameter Uncertainty	Able to capture parameter uncertainty.	None, deterministic values assumed for parameters.	None, deterministic values assumed for parameters.
Finding the Optimal Openness	Challenging due to the exhaustive nature of simulations.	Can identify optimal openness through iterative evaluations.	Identifies optimal openness within the four predefined scenarios.
Ease of Implementation	Requires details inputs and understanding to the simulation setup.	Moderate complexity, requires some knowledge the GLCC model.	Relatively easy to implement with simplified assumptions.

Table 6.1 presents a comparison of the three approaches. The GLCC predictor is advantageous during the early stages of architecture development, where limited information about the system is available. With minimal input requirements, it delivers a preliminary assessment, offering a broad indication of whether open or closed configurations are more suitable for the system architecture. As the development advances and more parameters are solidified, the GLCC becomes more applicable. This model excels at determining the openness of specific components and interfaces with growing accuracy. In the later stages of development, the simulation, demanding the most inputs, emerges as the most suitable approach. It enables a more precise and comprehensive setup, encompassing production schedules, diverse refresh scenarios, and deployment considerations, making it well-suited for evaluating the most sophisticated scenarios.

In this concluding chapter, we revisit the core questions outlined in Chapter 1, providing comprehensive insights based on our research findings:

1. What are the costs avoided and added due to openness?

Understanding the costs avoided and added associated with system openness is approached by considering components and interfaces separately. Open components like COTS

components initially incur lower development and procurement costs, yet due to their shorter support lives, they may become obsolete quickly, resulting in the need for more frequent refresh cycles throughout the life cycle. The shorter time to obsolescence of open components requires higher costs to sustain the post obsolescence period, such as increased bridge buy quantities and higher inventory holding costs. On the other hand, open interfaces have higher development costs but boast longer support lives compared to closed interfaces, potentially resulting in higher development costs but fewer refresh costs. Open interfaces enable "plug-and-play" functionality and mitigate the "ripple effect" during system refresh, minimizing the impact of a refreshed component and effectively reducing the number of components requiring refresh.

2. What are the variables that should be considered in assessing the cost impact associated with system openness?

When evaluating the cost impact associated with system openness, the primary variables considered for open/closed components/interfaces are the corresponding development cost, production cost, and support life. Additionally, various other parameters, such as refresh interval, the number of systems, bridge buy ratio, etc., play crucial roles in influencing the overall life-cycle cost and subsequently affect the selection of system openness. For instance, a more frequent refresh cycle may lead to increased overall redevelopment and reproduction of systems, providing an advantage to the components/interfaces with lower development and production costs. Similarly, a higher bridge buy ratio can result in the purchase of more spare components when a component becomes obsolete, favoring the closed components with longer support lives. Therefore, when assessing the cost impact associated with system openness, a comprehensive consideration of all these variables and parameters, whether directly linked to open/closed components/interfaces or not, is essential.

3. What level of openness provides the best value (largest cost avoidance) to the customer?

In this dissertation, a case study involving the A-RCI has been employed to illustrate the application of quantitative cost analysis in relation to system openness. Determining the optimal level of openness that consistently provides the best cost avoidance to customers is challenging, as various parameters and variables have the potential to impact the decision-making process for optimal openness. All three proposed approaches in this dissertation have been used on the A-RCI case, resulting in the calculation of life-cycle cost differences between the original A-RCI (as implemented by the US Navy) and a less open version (from simulation), as well as comparisons of relative life-cycle costs (from the GLCC model and predictor). The findings reveal that, from a life-cycle cost perspective, both open and closed systems can be less expensive based on specific parameter and variable settings. However, it is evident that a fully open architecture may not always represent the optimal solution. Rather than asserting that an open system consistently leads to cost-effectiveness, this study emphasizes the importance of conducting a comprehensive life-cycle cost analysis, considering various aspects of system openness.

4. What rule of thumb can be generalized based on the developed model?

To find the rule of thumbs for the selection of the system openness, this dissertation began with the simulation which tried to incorporate everything that is relevant to the life-cycle cost and system openness, adding more assumptions, removing the terms that is less relevant, and at the end came up with an analytic form that can connect with the system openness with life-cycle cost. The life-cycle cost of a component/interface can be concluded in Equation (6.1)

$$LCC = \left(DC + N_{sys}PC\right) + R\left(DC + N_{sys}PC(1 + M + B + H)\right) \tag{6.1}$$

Equation (6.1) integrates variables and parameters relevant to system openness and life-

cycle costs.

The first bracket represents the initial development and production of the component/interface. The second bracket is the recurring cost in each refresh cycle, multiplied by the discount factor R (sum of all recurring costs). The recurring cost includes the redevelopment and reproduction of the component/interface. Additionally, costs associated with maintenance (M), bridge buy (B), and spare holding (H) in each refresh cycle are considered, with their proportions tied to fleet size and procurement cost, subject to variations based on component support lives. Detailed insights into the development process of Equation (6.1) can be found in Chapter 5.

In conclusion, while Equation (6.1) may lack the sophistication of the simulation or the GLCC model, as it does not capture the intricate interactions within the architecture network, such as the ripple effect, it serves as a practical rule of thumb for comparing life-cycle costs between open and closed components/interfaces.

6.2 Future Work

The future work associated with the system openness can include:

1. Broadening the Definition of System Openness: The current working definition of system openness focuses primarily on the usage of open/closed component/interfaces, providing a straightforward basis for quantification analysis. However, there are other dimensions of system openness, such as modularity, that need comprehensive definition and translation into associated cost values. Future work should aim to expand the working definition to encompass these additional aspects, contributing to a more nuanced understanding of system openness.
2. Extending the Scope of System Openness Influence: While this dissertation concentrates on the joint impact of development/production costs and support lives on life-cycle costs,

there are other facets of system openness that remain unexplored. Elements like component reuse, vendor lock, system security, competition among independent sources, and intellectual property rights represent potential influences that warrant consideration in future analyses. A more comprehensive exploration of these factors can enhance our understanding of the broader implications of system openness.

3. Incorporating Uncertainty and Dynamics: The three approaches proposed in this study primarily operate within a statistical framework. While the simulation can integrate probabilistic models for component time to failure and obsolescence, the GLCC model and predictor predominantly utilize deterministic values, overlooking the uncertainty of life-cycle costs. Additionally, the approaches currently assume a static refresh strategy. Future research could delve into dynamic adaptations that account for uncertainties in life-cycle costs, offering a more responsive and realistic representation of systems and their components as they evolve over time.
4. Application to Different Industry Domains: The A-RCI case study serves as a demonstration within the scope of naval systems. Future endeavors could extend the application of the proposed approach to diverse industry domains, exploring how variations in architecture networks or different life-cycle lengths may influence the outcomes. Examining systems in different contexts will contribute to a more robust understanding of the generalizability and applicability of the proposed methodologies across various industries.

6.3 Dissertation Contributions

The contributions of this dissertation are:

1. A working definition for system openness that is applicable to business case.
2. An integrated design structure matrix approach for modeling system openness.

3. A discrete-event simulation framework to evaluate the life-cycle cost difference between two different system openness implementation scenarios.
4. The most complete and accurate data set for A-RCI cost modeling. As the cost information of A-RCI was unpublished to the public, this dissertation reverse-engineered the cost data of A-RCI including the development cost of the system architecture, the development and production cost of components, and the maintenance cost of the system.
5. A systematic method for assessing the ripple effect during system refresh for open system architecture. This approach identifies components and interfaces necessitating refresh in each review by considering their respective support lives and the system architecture network.
6. The GLCC model and the GLCC predictor, these groundbreaking methodologies represent the first attempts to establish a generalized relationship between life-cycle cost and system openness, particularly tailored to critical systems integrating hardware and software components.

Appendix A Supporting Data and Calibration of the A-RCI Case Study

In this appendix, a calibration of A-RCI data is performed based on a 2006 summary presentation from Advanced Systems Supportability Engineering Technologies and Tools (ASSETT) Corporation that summarizes top-down and bottom-up cost estimations for the A-RCI system. The information in this report was combined with other knowledge of the A-RCI system to construct a spreadsheet model of the A-RCI program activities and costs from 1996 through 2006. This spreadsheet model has in turn been used to calibrate the simulator developed in this program.

A simplified surrogate cost model on a spreadsheet is built based on the ASSETT report to calibration our simulation model. This model combines several input values from simulation model into a single input value in the spreadsheet model. For example, the value of production cost in Phase 1 represents the sum of procurement cost and installation fee of each component in the Phase 1 architecture. After the spreadsheet model was tuned to fit the known actual data, the input values of the simulation model can be evaluated based on the input values of spreadsheet model. A formulaic representation of this problem is,

$$f(n, x) = y \tag{A.1}$$

$$g(m, x) = y \tag{A.2}$$

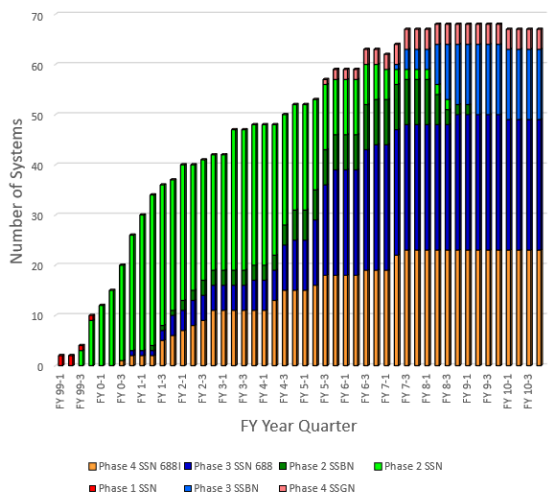
The function $f(.)$ represents the simulation model, and the function $g(.)$ is the spreadsheet model. y is the cost value, as a known output. x represents some known input variables (e.g., system production schedule) and n and m are some other unknown input variables (e.g., procurement cost per component). The relationship between n and m is known. The goal is to find n. The second equation is first solved and the values of m are determined. Then calculate the values of n using the relationship between n and m.

A.1 The Actual Data

The actual data for calibrating the simulation model input parameters included the production profile of the A-RCI and its life-cycle cost data. The production schedule and cost data are acquired from two different sources. While the production profile could be directly applied to the spreadsheet model, the cost data needed preprocessing.

A.1.1 The A-RCI Installation Profile

The installation profile (Schuster, 2007) in Figure A.1 shows the number of A-RCI systems fielded from FY99 to FY10. As shown in Figure A.1, the data is converted into three different tables, production schedule, retirement schedule, and the ship years[14] associated with each A-RCI phase. These three tables represent the known input variables x for the models in Equation (A.1) and (A.2). The production schedule and retirement schedule are used as the inputs of the simulation model. The production schedule and ship year data are used as the inputs of the spreadsheet model.



Production Schedule

FY	99	00	01	02	03	04	05	06	07	08	09	10
Phase 1	2	0	0	0	0	0	0	0	0	0	0	0
Phase 2	9	14	8	3	5	3	1	2	0	0	0	0
Phase 3	0	1	3	1	1	4	11	4	4	10	2	0
Phase 4	0	2	4	5	0	4	5	2	5	0	0	0

Retirement Schedule

FY	99	00	01	02	03	04	05	06	07	08	09	10
Phase 1	1	1	0	0	0	0	0	0	0	0	0	0
Phase 2	0	0	4	4	0	7	10	4	5	9	2	0
Phase 3	0	0	0	0	0	0	0	0	0	0	0	1
Phase 4	0	0	0	0	0	0	0	0	0	0	0	0

Ship year

FY	99	00	01	02	03	04	05	06	07	08	09	10
Phase 1	1.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Phase 2	3.00	17.25	28.50	27.25	29.75	28.25	22.25	17.25	12.25	6.50	0.50	0.00
Phase 3	0.00	0.25	2.00	4.75	5.25	7.75	15.50	22.75	27.25	34.25	40.50	40.00
Phase 4	0.00	0.75	3.75	8.75	11.00	13.50	17.50	21.00	25.50	27.00	27.00	27.00

Figure A.1 A-RCI production, retirements, and ship years by phase and year.

[14] Ship year represents the operation time of a fleet of ships. One ship year is equivalent to one ship operating for one year.

A.1.2 The Preprocessing of Life-Cycle Cost

The life-cycle cost data is obtained from ASSETT. This report compared the cost associated with two different submarine sonar systems: 1983 through 1993 traditional military standard model and 1996 through 2006 A-RCI model.

The cost data associated with submarine sonar from 1996 through 2006 was collected in the report. The ASSETT report used two approaches to obtain the cost: a top-down model from DoD budget allocations and bottom-up model from contract data. The two approaches used different cost categories. For the top-down model, the budget was split into four categories, RDT&E, SCN, OPN and O&MN.[15] In the bottom-up model, the total cost categories were: Development, Production-contractor, Production-government and O&S. In order to determine the needed data from the spreadsheet model result, three cost categories are defined that map the cost categories from both approaches. These categories are: Development cost, Production cost and O&S cost as shown in Figure A.2. The Development cost includes the RDT&E from top-down and Development from bottom-up model. The Production cost includes the SCN and OPN from top-down model and Production-contractor and Production-government from bottom-up model. The O&S cost includes O&MN from top-down model and O&S from bottom-up model.

[15] RDT&E includes "research, development, test and evaluation efforts performed by contractors and government." SCN is the "construction of new ships and conversion of existing ships." OPN represents "the procurement, production, and modernization of equipment" and O&MN is the "day-to-day costs of operating naval forces, including fuel, supplies, and maintenance of ships."

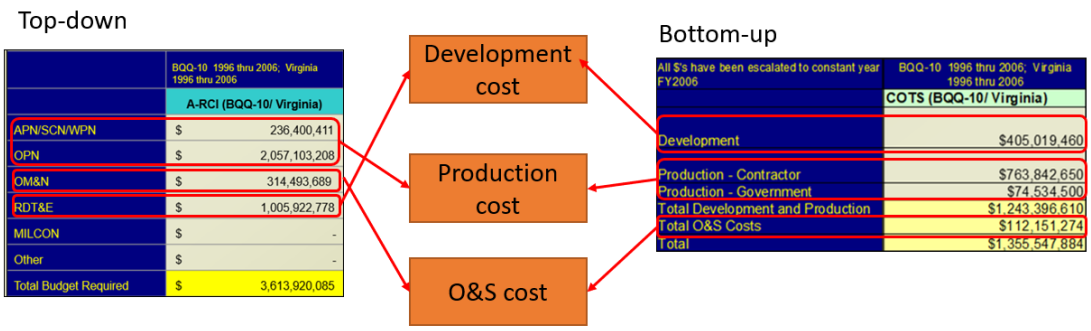


Figure A.2 ASSETT top-down and bottom-up cost model mapping.

The total life-cycle cost value from the two approaches were significantly different (see the Totals in Figure A.2). The total life-cycle cost from top-down model is approximately three times the total life-cycle cost from bottom-up model. According to McCaig (2020), the top-down budget allocation was estimated based on an older and more conservative cost model that did not appropriately reflect the cost savings from the COTS-based acoustic system. Thus, the bottom-up life-cycle cost was more appropriate to represent the true cost.[16]

In the summary presentation of the ASSETT report, the yearly budget values were provided, while only the total cost from bottom-up approach was shown in the report. The yearly bottom-up cost is estimated based on the distribution of annual budget. In the right side of Figure A.3, the blue curve is the annual top-down budget and the red-dashed curve is the estimated annual bottom-up cost, which is regarded as the true value of life-cycle cost. Using the true cost values of each categories, the next step is to tune our spreadsheet model to make the annual cost result as close as the bottom-up curve as possible.

[16] Additionally, the top-down approach for the A-RCI model represents the appropriations approved by congress for the R&D, production, and support of the A-RCI system over the ten-year period from 1996 to 2006. These appropriations are funding requests put together by the Navy based on their then cost models and estimates. The bottom-up approach represents the funding awarded in contracts from 1996 to 2006 for R&D, production and support. Therefore, the bottoms-up approach is the true cost for the ten-year period.

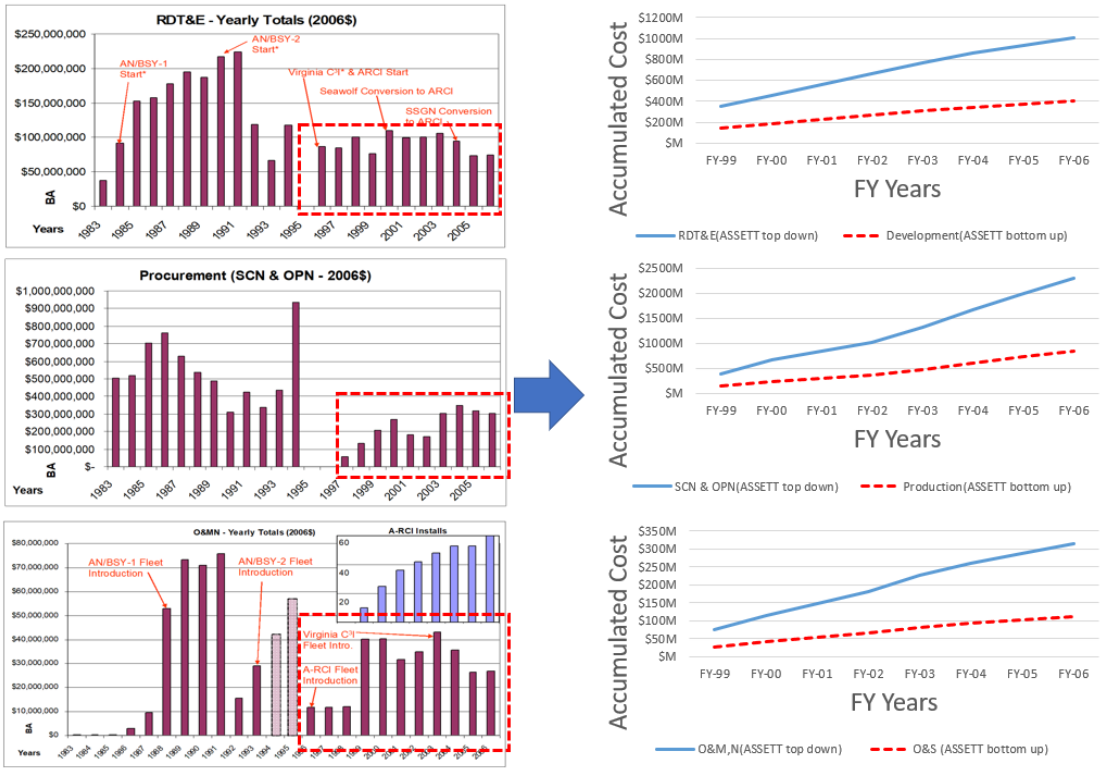


Figure A.3 Annual top-down (ASSETT, 2006) (left) and estimated annual bottom-up cost.

A.2 The Spreadsheet Model

The spreadsheet model contains 5 input parameters, which are listed in Table A.1. The value of each parameter affects one of the three cost categories: Development cost, Production cost and O&S cost.

Table A.1 The Input Parameters of the Spreadsheet Model

Cost categories	Development		Production		O&S
Input parameters	Average development cost of each phase	APB development cost (per year)	Average production cost of each phase (per system)	APB production cost (per system)	O&S cost per ship per year
		TI development cost (per year)		TI production cost (per system)	

A.2.1 The Development Cost

The Development cost includes the development effort of the four A-RCI phases and the development effort of refresh, Technology Insertion (TI) and Advanced Processing Build (APB). The development of the four phases was probably accomplished over the span of several years before production. Those years are combined and assumed it all occurred in year

1.[17] The development of APB and TI occurred annually and bi-annually respectively. It is assumed that the development of APB began in year 3 and the development of TI began on year 4. The entire development schedule is provided in Table A.2. The total Development cost is the sum of the number of developments of each phase or refreshes multiplied by their associated development cost.

Table A.2 A-RCI Development Schedule

Year	1	2	3	4	5	6	7
Phase 1 development	✓						
Phase 2 development	✓						
Phase 3 development		✓					
Phase 4 development		✓					
APB development			✓	✓	✓	✓	✓
TI development				✓		✓	

For the schedule shown in Table A.2, the APB installation was not available before year 3 (TI before year 4) since the refresh was not yet developed. In addition, the version of TI installation in year 4 would be the same as the installation in year 5.

A.2.2 The Production Cost

The production activity includes the production of the systems and the demand for system refresh. The production schedule of each phase is based on Table A.1Figure A.1 A-RCI production, retirements, and ship years by phase and , while additional assumptions needed to be made for the APB/TI installation schedule. Each submarine received a TI refresh every four years and an APB every two years (Johnson, 2004). It is assumed that only the systems in phase 3 and 4 received the APB and TI refreshes and that they began to receive refreshes one year after their production if the refresh is available. For example, if a phase 3 A-RCI is produced

[17] Note, this assumption has no impact on the spreadsheet since the spreadsheet does assume any discounting of the costs.

in year 3, it would receive its first APB in year 4 and second APB in year 6 and its first TI would also be in year 4 and second TI in year 8. Table A.3 shows the production/refresh installation number of submarines in each year. The total Production cost is the sum of the number of installations of each phase or refreshes multiplied by the associate production cost.

Table A.3 A-RCI Installation Schedule

Year	1	2	3	4	5	6	7
Phase 1 production	2	0	0	0	0	0	0
Phase 2 production	9	14	8	3	5	3	1
Phase 3 production	0	1	3	1	1	4	11
Phase 4 production	0	2	4	5	0	4	5
APB installation	0	0	3	7	9	8	17
TI installation	0	0	0	7	6	1	8

Unit: number of installations

A.2.3 The O&S Cost

For O&S cost, it is assumed that the O&S cost per ship year is the same in each phase. The number of ship years in each phase in each year, as shown in Table A.4, is calculated based on the production schedule. The total O&S cost is the product of the O&S post per ship year and the total number of ship years in the field.

Table A.4 The Number of Ship Years in each Phase in each Year

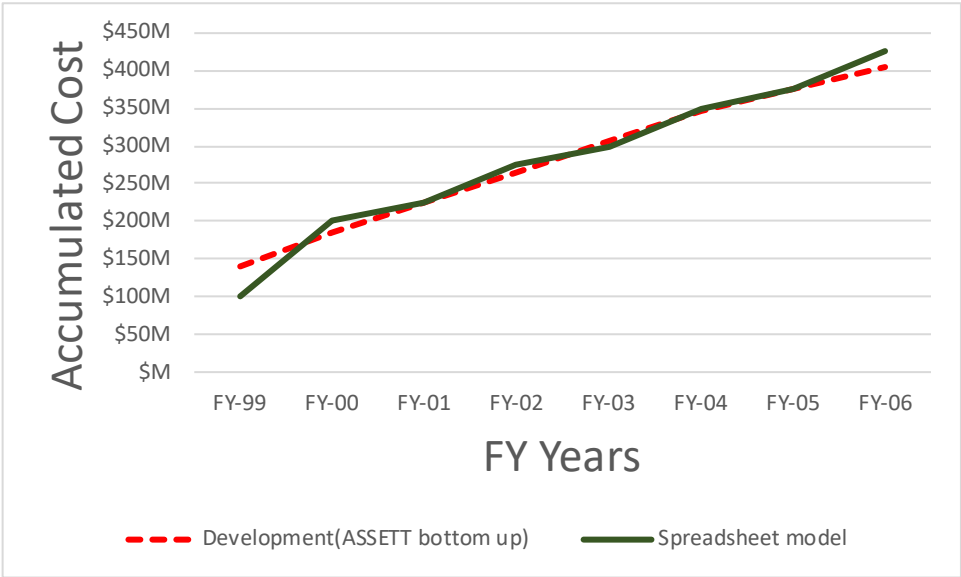
Year	1	2	3	4	5	6	7
Phase 1 Ship Year	1.50	0.00	0.00	0.00	0.00	0.00	0.00
Phase 2 Ship Year	3.00	17.25	28.50	27.25	29.75	28.25	22.25
Phase 3 Ship Year	0.00	0.25	2.00	4.75	5.25	7.75	15.50
Phase 4 Ship Year	0.00	0.75	3.75	8.75	11.00	13.50	17.50

Total Ship Year	4.50	18.25	34.25	40.75	46.00	49.50	55.25

Unit: ship years

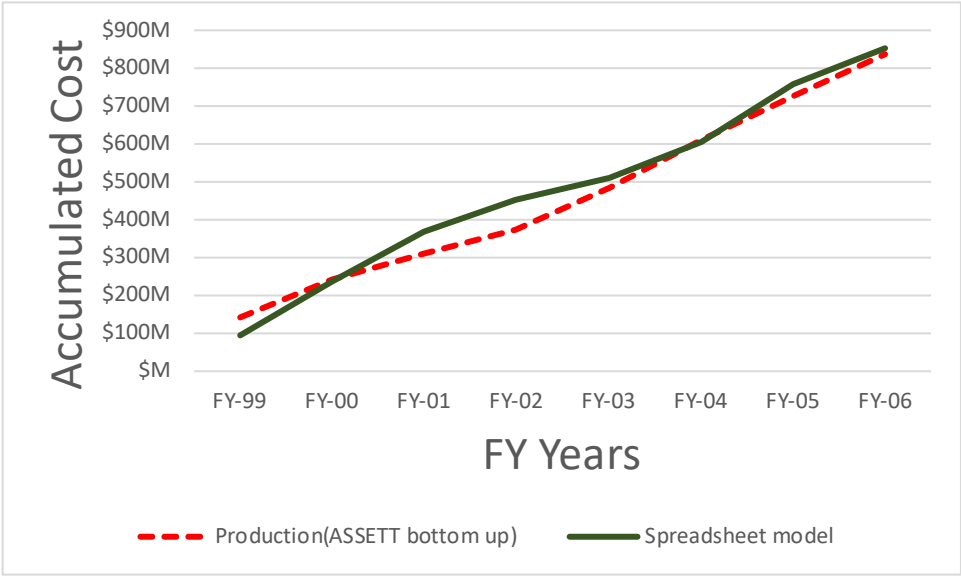
A.3 Spreadsheet Model Tuning Result

The 5 input parameters in the spreadsheet model are tuned to make the result fit the true life-cycle cost curve. Figure A.4 through A. 6 show the input parameters and the corresponding cost category result curve.



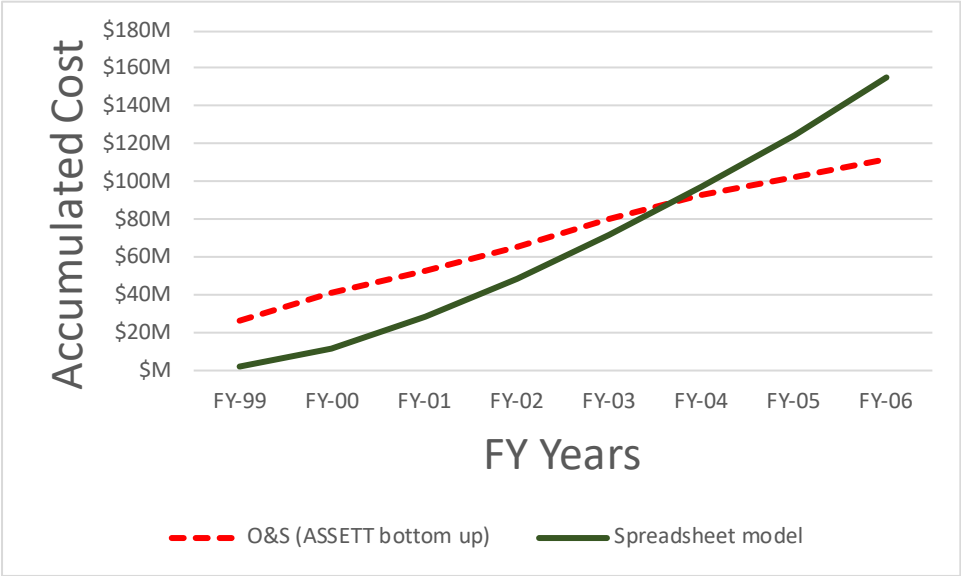
Average development cost of each phase		$50,000,000	
APB development cost	$25,000,000	TI development cost	$25,000,000

Figure A.4 The spreadsheet result of development cost and the tuned input parameters.



Average production cost of each phase per system		$8,500,000	
APB production cost per system	$100,000	TI production cost per system	$1,000,000

Figure A.5 The spreadsheet result of production cost and the tuned input parameters.



Unit level O&S cost per ship per year	$502,920

Figure A.6 The spreadsheet result of O&S cost and the tuned input parameters.

A.4 The Input Parameters of Simulation Model

Using the tuned input parameters from spreadsheet model, the input parameters for the simulation model are determined based on the assumed relationships. The results are shown in

Table A.5.

Table A.5 The Adjusted Input Parameters of the Simulation Model

Refresh development Cost per refreshable HW component	$2,083,333
Refresh development Cost per refreshable SW component	$12,500,000
Production Cost per refreshable HW component	$8,065
Production Cost per refreshable SW component	$100,000
Production Cost of infrastructure per system	$12,073,387
Installation/Procurement ratio	0.1
Procurement cost per refreshable HW	$7,331
Procurement cost per refreshable SW	$90,909
Procurement Cost per unit infrastructure	$671,087
Average number of O&S events per ship year	11
Cost per O&S event	$45,720
Maintenance action cost	$38,389

A.4.1 Development Cost per Refreshable Hardware/Software Component

For the parameters that are associated with refresh development, the refresh development cost per software/hardware component is the per APB/TI development cost divided by the total number of types of software/hardware components that need to be refreshed in each APB/TI. In each APB/TI, there were 12 types of hardware components and 2 types of software components needing refresh giving,

$$Refresh\ development\ Cost\ per\ refreshable\ SW\ component = \frac{\text{APB development cost}}{2} \tag{A.3}$$

$$Refresh\ development\ Cost\ per\ refreshable\ HW\ component = \frac{\text{TI development cost}}{12} \tag{A.4}$$

A.4.2 Production Cost per Refreshable Hardware/Software Component

It is assumed that the production cost (*PC*) per A-RCI system was the sum of the production cost of infrastructure, refreshable hardware and refreshable software. The production cost of all of the refreshable hardware in an A-RCI system is approximately the "TI production cost per system" and the production cost of all the refreshable software in an A-RCI system is approximately the "APB production cost per system."

$$PC_{ARCI} = PC_{infrastructure} + PC_{refreshable\ HW} + PC_{refreshable\ SW} \tag{A.5}$$

The production cost per refreshable hardware component are calculated by dividing the "TI production cost per system" (i.e., the production cost for all the whole refreshable hardware per system) by the total number of refreshable hardware components per system. In the equation below, the value 248 is equal to the total number of refreshable hardware components in phases 3 and 4. The production cost per refreshable software was the "APB production cost per system" (i.e., the production cost of the all the refreshable software) since it is assumed there was only one software component in each A-RCI system.

$$PC_{per\ refreshable\ HW\ component} = \frac{\text{2TI production cost per system}}{248} \tag{A.6}$$

$$PC_{per\ refreshable\ SW\ component} = \frac{\text{APB production cost per system}}{1} \tag{A.7}$$

Base on the description in (Guertin and Miller, 1998), the number of total refreshable hardware/software components of each A-RCI system in different phases is estimated. In the timeframe of the cost data, there were total 76 A-RCI systems produced, which also indicates that there were total 6488 refreshable hardware components and 76 refreshable software components produced among the 76 A-RCI systems. Once the production cost per refreshable hardware/software component is obtained, the production cost of the all the refreshable

hardware and software of each A-RCI system in different phases can then be calculated. The total production cost of the total A-RCI systems is also known from the bottom-up cost data. The total infrastructure production cost of the total A-RCI systems is the refreshable hardware/software production cost of all the A-RCI systems subtracted from the total production cost. Assuming that the infrastructure production cost is approximately the same for each system, the production cost per system infrastructure is then the total infrastructure production cost of all the A-RCI systems divided by the total number of A-RCI systems.

$$\begin{aligned} PC_{total} &= N_{system} PC_{infrastructure\ per\ system} \\ &\quad + N_{total\ \text{refreshable HW component}} \text{PC}_{\text{per refreshable HW component}} \\ &\quad + N_{total\ \text{refreshable SW component}} \text{PC}_{\text{per refreshable SW component}} \\ &= 76 PC_{infrastructure\ per\ system} + 6488 \text{PC}_{\text{per refreshable HW component}} \\ &\quad + 76 \text{PC}_{\text{per refreshable SW component}} \end{aligned} \tag{A.8}$$

The production cost per component can be separated into the component procurement cost and installation cost, which can be calculated based on the assumption that the installation cost is one tenth of the procurement cost.

$$\begin{aligned} PC_{per\ refreshable\ HW\ component} &= \text{ProcurementCost}_{\text{per refreshable HW component}} \\ &\quad + \text{InstallationCost}_{\text{per refreshable HW component}} \end{aligned} \tag{A.9}$$

A.4.3 Maintenance Action Cost

According to the ASSETT report, there were total 573 O&S events among 52 ships in 2006. It is inferred that there are average of 11 O&S events in one ship year. The cost per O&S event is the O&S cost per ship year divided by the number of O&S events per ship year.

Assuming that each O&S event occurred when there was a component failure. The cost per

O&S event includes the procurement cost of a new component to replace the failed one and other maintenance actions. Therefore, the cost of maintenance actions is the cost per O&S event subtracted from the procurement cost. Assuming that all the failures are caused by the refreshable hardware components, the procurement cost of the refreshable hardware components was used to calculate the maintenance action cost.

$$O\&SCost_{per\ event} = \text{ProcurementCost}_{\text{per refreshable HW component}} + Cost_{maintenance} \quad (A.10)$$

A.5 The Resulting Simulation Model

The simulation input parameters were modified according to the process described in the previous section. Figure A.7 shows the total life-cycle cost results of simulation model and the true total life-cycle cost based on bottom-up data.

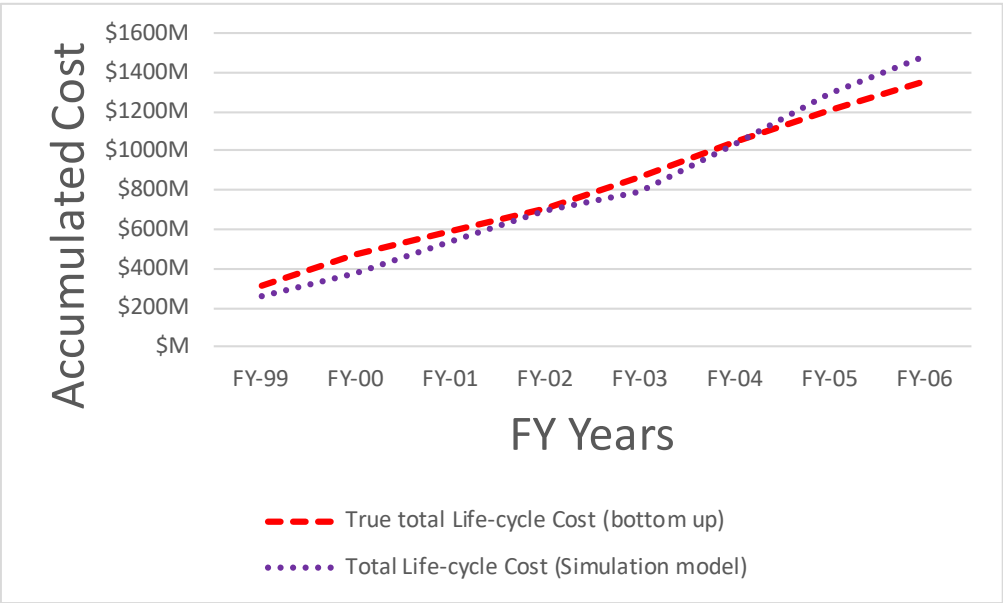


Figure A.7 The cost result comparison of simulation model and spreadsheet model.

Figure A.7 shows that the simulation model is adjusted to produce similar life-cycle cost to the actual data. Given that the input variables of: installation schedule (Schuster), refresh

schedule (Johnson, 2004), system configuration (Guertin and Miller 1998), etc., and assuming that the simulation model is reasonable, it can be concluded that the unknown input cost parameters are properly tuned.

Noted that in this simulation result, there are some costs that have negligible contributions (in the context of the A-RCI case considered) and have not been considered due to the current refresh setting and the short timeframe. For example, the cost from component obsolescence is neglectable because the rapid refresh schedule assumed. The effect of the obsolescence of standards is also not included because the effective lifetime of a standard is usually longer than 10 years. These costs may play more important roles through the longer system life cycle and for alternative assumptions about management of the system.

References

Abbott, J. W., Levine, A., & Vasilakos, J. (2008). Modular/open systems to support ship acquisition strategies. *Proceedings of the American Society of Naval Engineers Day*, Arlington, VA.

Abts, C. (2002). COTS based systems (CBS) functional density - A heuristic for better CBS design. In COTS-Based Software Systems (pp. 1-9). Orlando, FL: Springer.

Abts, C. M., & Boehm, B. W. (1997). COTS software integration cost modeling study. *Contract, 30602(94-C), 1095*.

Abts, C., Boehm, B. W., & Clark, E. B. (2000). COCOTS: A COTS software integration lifecycle cost model-model overview and preliminary data collection findings. In *Proceedings of the ESCOM-SCOPE Conference* (pp. 18-20).

ASSET. (2006, August 25). *Submarine sonar cost analysis report (Rev. 10)* [PowerPoint slides].

Bass, L., de Niz, D., Hansson, J., Hudak, J., Feiler, P. H., Firesmith, D., ... & Wallnau, K. (2008). Results of SEI independent research and development projects. *Software Engineering Institute, Carnegie Mellon University* (Technical Report CMU/SEI-2008-TR-017).

Boudreau, M. (2006). *Acoustic rapid COTS insertion: A case study in spiral development*. Naval Postgraduate School Report.

Clark, B., & Clark, B. (2007). Added source of costs in maintaining COTS-intensive systems. *Cross Talk, The Journal of Defense Software Engineering, 20*(6), 4-8.

Colombi, J. M., Robbins, M. J., Burger, J. A., & Weber, Y. S. (2015). Interface evaluation for

open system architectures using multi-objective decision analysis. *Military Operations Research, 20*(2), 55-69.

Dargan, P. A. (2005). *Open systems and standards for software product development*. Norwood, MA: Artech House.

Davendralingam, N., Guariniello, C., Tamaskar, S., DeLaurentis, D., & Kerman, M. (2019). Modularity research to guide MOSA implementation. *The Journal of Defense Modeling and Simulation, 16*(4), 389-401.

Defense Standardization Program. (2002). What is MOSA. Retrieved December 22, 2020, from https://www.dote.osd.mil/Publications/Annual-Reports/2002-Annual-Report/

Deng, G. (2007). *Simulation-based optimization* (Doctoral dissertation, University of Wisconsin--Madison).

DoD. (2015). *DoD instruction 5000.02, operation of the defense acquisition system*. Retrieved December 22, 2020, from https://www.acq.osd.mil/fo/docs/500002p.pdf

DoDAF. (n.d.). DM2 data groups. Retrieved from https://dodcio.defense.gov/Library/DoD-Architecture-Framework/dodaf20_capability_mm/#:~:text=A%20capability%2C%20as%20defined%20here,published%20by%20the%20Joint%20Staff

Dunn, L. M. (2011). An investigation of the factors affecting the lifecycle costs of COTS-based systems (Doctoral dissertation, University of Portsmouth).

Dunn, S., Laroche, J., & Mitchell, P. (2018). Open system architectures for the ADF: Opportunities and challenges. *Australian Defence Force Journal, 204*.

Eppinger, S. D., & Browning, T. R. (2016). *Design structure matrix methods and applications*. Cambridge, MA: The MIT Press.

Erdogmus, H., Sledge, C. A., Allison, S., & Looney, M. (2000, June). Costing of COTS-based systems: An initial framework. In *Proceedings of the 2000 Workshop on Component-Based Software Engineering*, Limerick, Ireland.

Firesmith, D. (2015, October 5). Open system architectures: When and where to be closed. *Software Engineering Institute Blog*. Retrieved January 5, 2021, from https://insights.sei.cmu.edu/sei_blog/2015/10/open-system-architecture-when-and-where-to-be-closed.html

GAO. (2014). Review of private industry and department of defense open systems experiences. *GAO-14-617R*.

Guertin, N. H., & Miller, R. W. (1998). A-RCI -- The right way to submarine superiority. *Naval Engineers Journal, 110*(2), 21-33.

Hanratty, J. M., Lightsey, R. H., & Larson, A. G. (2002). Open systems and the systems engineering process. *Office of the Undersecretary of Defense, Acquisition and Technology; Open Systems Joint Task Force*.

Henderson, P. (2009, December 14). The case for open systems architecture. Retrieved from http://pmh-systems.co.uk/Papers/MOSACaseFor/

Hosking, R. (2020). SOSA and VITA: Working together for next-gen defense systems. *VITA: Embedded Tech Insights*. Retrieved December 21, 2023, from https://vita.militaryembedded.com/6902-sosa-and-vita-working-together-for-next-gen-defense-systems/

Jensen, F., & Petersen, N. E. (1982). *Burn-in: An engineering approach to the design and analysis of burn-in procedures*. New York: Wiley.

Johnson, W. (2004). The A-RCI process—Leadership and management principles. *Naval*

Engineers Journal, 116(4), 99-106.

Kowalski, N. W., Oblinger Jr, J. T., & Peresta, W. J. (1998, April). Open systems engineering effectiveness measurement. In *Software Technology Conference* (Vol. 21).

Lebron Jr, R. A., Rossi, R., & Foor, W. (2000). Risk-based COTS systems engineering assessment model: A systems engineering management tool and assessment methodology to cope with the risk of commercial off-the-shelf (COTS) technology insertion during the system life cycle. In *Strategies to Mitigate Obsolescence in Defense Systems Using Commercial Components* (Vol. 7, pp. 1).

Leffet, H., Ben Sassi, S., & Ben Ghezala, H. (2011, December). Cost estimation model for COTS-based projects. In *Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference* (pp. 258-265).

Lewis, P., Hyle, P., Parrington, M., Clark, E., Boehm, B., Abts, C., & Manners, R. (2000). Lessons learned in developing commercial off-the-shelf (COTS) intensive software systems. *Federal Aviation Administration Software Engineering Resource Center Report*.

Logan, G. T. (2004). The modular open systems approach (MOSA). *OSJTF presentation to the Executive Program Managers Course*. Retrieved from acc.dau.mil/CommunityBrowser.aspx?id=37585

McCaig, B. (2020, July 12-13). [Communication with researchers]. Washington, DC.

Minkiewicz, A. F. (2001, March). The real costs of COTS. In *2001 IEEE Aerospace Conference Proceedings* (Vol. 6, pp. 2863-2869). IEEE.

MOSA. (2012). AFRL/RYM Metrics Working Group, MOSA Metrics Calculator. *Unpublished*.

NOAET. (2009). Naval Open Architecture Enterprise Team, Open Architecture Assessment

Tool, Version 3.0, User's Guide.

Oberndorf, P. (1998). COTS and open systems. *SEI Monographs on the Use of Commercial Software in Government Systems, 11*.

Office of the Deputy Director for Engineering & Office of the Under Secretary of Defense for Research and Engineering. (2021). *Systems engineering guidebook*. Washington, D.C.

Office Secretaries of the Navy, Army, and Air Force. (2019, January 7). Modular open systems approaches for our weapon systems is a warfighting imperative [Memorandum].

OSJTF. (2004). Open Systems Joint Task Force, "Program Manager's Guide: A Modular Open Systems Approach (MOSA) to Acquisition," United States Department of Defense. Retrieved September 2004, from http://www.acq.osd.mil/osjtf/pdf/PMG_04.pdf

Paul, J. T. (1998, October). COTS based open systems for military avionics. In *17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings* (Vol. 2, pp. G31-1). IEEE.

Peruzzi, L. (2019). A new generation of submarine combat management systems. *European Security and Defence*. Retrieved December 29, 2020, from https://euro-sd.com/2019/05/articles/13130/a-new-generation-of-submarine-combat-management-systems/

Sandborn, P. (2017). *Cost analysis of electronic systems* (2nd ed.). World Scientific.

Schramm, Z. (2013). A model for estimating the cost tradeoffs associated with open electronic systems (Master's thesis, University of Maryland).

Schuster, J. (2007). Recent progress in submarine sonar … and future challenges in ASW. *MIT Department of Mechanical Engineering, Center for Ocean Engineering, Ocean Acoustics Group*. Retrieved from

http://acoustics.mit.edu/dyerparty/Ira%20Dyer%20talk%20rev%201.pdf

Shepherd, K., & Wills, J. (2018). Avionics open systems architecture standardization.

Tamaskar, S., Neema, K., & DeLaurentis, D. (2014). Framework for measuring complexity of aerospace systems. *Research in Engineering Design, 25*, 125-137.

UK MoD. (2013). Guidance system of systems approach (SOSA) open systems strategy. Retrieved December 23, 2020, from https://www.gov.uk/government/publications/system-of-systems-approach-sosa-open-systems-strategy

Wright, M., Humphrey, D., & McCluskey, P. (1997). Uprating electronic components for use outside their temperature specification limits. *IEEE Transactions on Components, Packaging, and Manufacturing Technology, Part A, 20*(2), 252-256.

Yang, L. (2005). Generalization of an integrated cost model and extensions to COTS, PLE and TTM (Doctoral dissertation, West Virginia University).