Analysis of the life-cycle cost and capability trade-offs associated with the procurement and sustainment of open systems

Shao-Peng Chen and Peter Sandborn*

Department of Mechanical Engineering,
University of Maryland,
College Park, MD 20742, USA
Email: spchen@umd.edu
Email: sandborn@umd.edu
*Corresponding author

William Lucyshyn

Center for Governance of Technology and Systems,
School of Public Policy,
University of Maryland,
College Park, MD 20742, USA
Email: lucyshyn@umd.edu

Abstract: System openness refers to the extent to which system components (e.g., hardware and software) can be independently integrated, removed, or replaced without adversely impacting the existing system. Openness is an intuitively understood concept used to describe the architecture and implementation of safety-, mission- and infrastructure-critical systems. While openness is widely associated with life-cycle cost avoidance, system openness can also lead to increased costs in some cases. Previous efforts to establish value have relied on qualitative system analyses, with the results often articulated as an intangible 'openness score' that fails to provide the information necessary to understand the conditions under which there is a life-cycle cost avoidance. This paper develops a model that quantifies the relationship between system openness, life-cycle cost and system capability risks. A case study that evaluates the acoustic rapid COTS insertion (A-RCI) Sonar System is provided.

Keywords: open systems; life-cycle cost modelling; capability; obsolescence management; acoustic rapid COTS insertion; A-RCI; open systems approach; OSA; system of systems approach; SOSA; business case.

Reference to this paper should be made as follows: Chen, S-P., Sandborn, P. and Lucyshyn, W. (2022) 'Analysis of the life-cycle cost and capability trade-offs associated with the procurement and sustainment of open systems', *Int. J. Product Lifecycle Management*, Vol. 14, No. 1, pp.40–69.

Biographical notes: Shao-Peng Chen is a PhD student in the Reliability Engineering Program at the University of Maryland advised by Dr. Peter Sandborn. He received his BS degree in Mechanical Engineering from National Taiwan University and had a two-year experience as a Mechanical Design Engineer in the Robotic Industry. His current research is

focused on modelling and quantifying the life-cycle cost and benefits of open systems. He is interested in strategizing system maintenance, including constructing reliability models based on real-world data and implementing the models for a simulation-based optimisation to improve current maintenance practices.

Peter Sandborn is a Professor in the Department of Mechanical Engineering at the University of Maryland. His research group develops models for optimising the sustainment of critical systems. He is an Associate Editor of the *IEEE Transactions on Electronics Packaging Manufacturing* and a member of the Board of Directors of the PHM Society. He is a fellow of the IEEE, the ASME and the PHM Society. He is the author of several books on cost modelling, maintenance, and technology obsolescence and over 200 technical papers.

William Lucyshyn is a Research Professor and the Director of Research at the Center for Governance of Technology and Systems, in the School of Public Policy, at the University of Maryland. His current projects include public and private sector partnering; transforming Department of Defense sustainment and supply chain management; and identifying government sourcing and acquisition best practices. Previously, he served as a SES Program Manager and the Principal Technical Advisor to the Director of the DARPA and currently serves on the editorial board for the *Defense Acquisition Research Journal*.

This paper is a revised and expanded version of a paper entitled 'Understanding and modeling the life-cycle cost tradeoffs associated with the procurement of open systems' presented at Acquisition Research Symposium, Monterey, California, USA, 13–14 May 2020.

1 Introduction

Manufacturers and sustainers of critical systems face many long-term budgetary challenges. Critical systems, such as aircraft, rail, industrial controls, power generation, defence and communications infrastructure, are expensive to procure and sustain over their long-life cycles (measured in decades). Because of their prohibitively high cost of replacement (which is in many cases borne by taxpayers), these safety-, mission-, and infrastructure-critical systems cannot be replaced as often as they should be. Concurrently, technology is evolving, which limits the useful life of many systems, thus requiring frequent upgrades to maintain the capability that is required to remain competitive or effective in accomplishing their intended purpose.

Critical systems have traditionally been developed using acquired proprietary systems and interfaces, which make it challenging to modernise and reduces opportunities for competition. Implementing an open-systems architecture permits the development and acquisition of modular, interoperable systems enabling components to be added, modified, or replaced by different vendors throughout the system's life cycle. This creates the potential for increased competition and innovation (Guertin and Womble, 2012). For example, the US Air Force's program to upgrade the B-2 bomber's communications, networking, and defensive management systems will cost over $2 billion, in part because the prime contractor owns all the necessary proprietary

technical data and software. Because this system is 'closed', competing this effort was not a viable economic option (GAO, 2014).

A variety of strategies are being explored, or reemphasised, to increase the efficiencies of acquisition processes. One way for the critical systems community to minimise 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 can provide a degree of flexibility, enabling the integration of rapidly changing technologies. However, as with all approaches, there are costs as well as benefits. This paper explores a business case methodology to assess the application-specific cost effectiveness of OSA.

1.1 Open systems approach

System openness refers to the extent to which system components (e.g., hardware and software) can be independently integrated, removed, or replaced without causing an adverse impact on the existing system. The quality, functionality, and efficiency of open-system architectures have led to their wide acceptance by business and industry, and they are starting to be used by nations that have advanced defence industries (Mladenović et al., 2013).

For example, the US 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 OSA, now referred to as modular open systems approach (MOSA),[1] promotes the use of modular design to encourage companies to improve and manufacture technologies that are interoperable with the DoD's current system.

The UK's Ministry of Defence (MoD) has a similar strategy for the use of open-systems architecture referred to 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 defence enterprise. The MoD's policy provides high-level guidance and states that an open-systems approach should be adopted to realise 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 commercial off the shelf (COTS) 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).

Conventional wisdom supports the notion of open systems but quantifying the actual cost avoidance remains elusive. The objective of this paper is to create a model that allows the quantification of the relationship between system openness and life-cycle cost on an application-specific basis. An example of this could be the use of radar technology on an aircraft. With an OSA, the radar technology could be replaced or upgraded without replacing numerous related subsystems. Closed-systems architecture, on the other hand, effectively restricts access to configuration and programming information from outside parties. Closed systems often make upgrading a piece of equipment difficult and costly.

Further, closed systems can lead to 'vendor lock' where the customer becomes dependent on a single service provider because the costs of changing vendors are prohibitive.

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 generalisability 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).

While the defence community supports implementing OSA whenever possible, there are numerous reasons to be cautious since business and engineering-trade-offs must be made that could change the incentive structure and reduce the system effectiveness. First, if there are no standards for a new product, then a closed-system architecture may be best until standards are created (Firesmith, 2015). Second, if there is only one vendor that can provide a subsystem or service (i.e., a 'sole source'), then attempting to make an open system may have no benefit; for example, the remote vision system on the Boeing KC-46A[2] is sole sourced to Boeing (SAF Public Affairs, 2020). This newly developed complex system is critical to the aircraft's primary mission (aerial refuelling), it must be fully integrated with the proprietary aircraft systems and has limited/no application to other military systems or commercial use. As a result, a sole source development was the most efficient solution. Finally, the system support duration and the quantity of systems supported have to be carefully considered in the open versus closed system design decision (e.g., see the A-RCI case study presented in Section 3 of this paper).

Generally, it is implicitly assumed that the use of OSA decreases the total life-cycle cost of a system. Leveraging existing open technology, including COTS components, avoids many costs associated with designing custom systems, and reduces the time required for the development or refresh of a system (Logan, 2004). The use of OSA 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. The use of well-defined standards promotes smooth interfacing both within and between systems, while the proliferation of common components has the positive impact of fostering competition between suppliers. Component design reuse (within and between systems) eliminates redundant components, thus reducing logistical costs.

However, there are costs associated with openness that should be considered. Building a subsystem from commercially available components might add unnecessary functionality into the system and increase the system complexity, resulting additional effort for component and system-level qualification (Grant et al., 2000; Hanratty et al., 2002; Clough, 2003). Alternatively, 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 may make it necessary 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.

This paper seeks to quantitatively analyse OSA, specifically the relationship between OSA and life-cycle cost.

1.2 Existing work

Several previous efforts have addressed the measurement of system openness. The MOSA Program Assessment and Rating Tool (PART or MOSA PART) was developed by the US Navy's Open Systems Joint Task Force (OSJTF) (OSJTF, 2004). Based upon MOSA PART, the Naval Open Architecture Enterprise Team (OAET) developed the Open Architecture Assessment Model (OAAM) and the Open Architecture Assessment Tool (OAAT) (NOAET, 2009). These tools used similar user-based ratings to measure a system's openness. Tool users first answer a series of self-rating questions. Based on the weights of each question, the tools then calculate the final ratings for system openness principles. The tools primarily ask system-level and 'to what extent' questions, e.g., 'to what extent do system components and selected commercial products conform to standards selected for system interfaces?'. The answers to these qualitative questions are highly dependent on the users' subjective points of view towards the system and program.

In 2011 and 2012, several parties including the US Air Force Research Laboratory's RYM subgroup collaborated to develop a set of metrics to evaluate the openness of an architecture. This effort resulted in a new tool called the MOSA metrics calculator (MOSA, 2012). Instead of asking subjective system-level questions, the MOSA metrics calculator uses objective component-level questions. The component characteristics that result from the MOSA metrics calculator are more quantifiable and can be accumulated.

In addition to the problem of subjectivity, a common problem that these tools share is that they were not designed to assess the cost associated with openness. The main goal of PART, OAAT, and the MOSA metrics calculator is measuring the level of conformance to the open system principles, while assuming that increased openness is always beneficial. Without accounting for cost in detail, assuming that the value of benefits outweighs the costs in every case is questionable.

Another approach to measuring openness comes from PMH Systems and the University of Southampton. This work uses a 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 also 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 life-cycle-cost models of COTS-based systems have been proposed by several researchers. COTS integration cost calculator (CICC) and constructive COTS cost model (COCOTS) estimate the costs incurred to integrate a COTS software into a larger system (Abts and Boehm, 1997; Abts et al., 2000). These proposed models are limited to the impact of COTS software implementations.

Previous efforts either rely on a highly qualitative analysis of a system to compare system implementations to determine openness, or are limited to specific systems and partial system openness, e.g., the evaluation of COTS in software systems. These approaches do not provide sufficient information to understand the conditions under

which life-cycle cost avoidance can be maximised (or whether there even is cost avoidance), or to make business case decisions. The objective of this paper is to quantify the relationship between system openness and life-cycle cost.

Section 2 of this paper describes a stochastic discrete-event simulation model developed to determine the difference in life-cycle and implementation cost between two versions of the same system (having different levels of openness). Section 2 also introduces a model to quantify the value of capability updates to the system. Section 3 provides a case study using the model. In Section 4 we discuss the generalisation of the model.

2 Model development

In this study, our goal is to compare the life-cycle cost between two different system configurations, determining which one is more cost beneficial. This section presents a stochastic discrete-event simulation developed to generate a list of system life-cycle events that were then used as the inputs to a cost model. The cost model is then used to determine the difference in life-cycle cost between two versions of the same system (the versions having different levels of openness). A demonstration version of this process is provided in the Appendix.

The total life-cycle cost incurred designing, building, operating, and retiring a system is,[3]

$$C_{Total} = C_{Development} + C_{Production} + C_{O\&S} + C_{Refresh} + C_{Capability} \quad (1)$$

where $C_{Development}$ are the non-recurring costs of system design, development and qualification; $C_{Production}$ captures the recurring costs to manufacture and field the system; $C_{O\&S}$ are the costs of sustainment incurred from system deployment to the end-of-support; and $C_{Refresh}$ are the costs of implementing and qualifying technology refresh(es) during the system support life. $C_{Capability}$ are the potential costs associated with the technology in the system being out-of-date relative to the state-of-practice.

A discrete-event simulation was developed as the first step of the life-cycle cost comparison process in this paper. In general, discrete-event simulators model a sequence of events along a timeline. A deterministic schedule or probability distributions are used as inputs to the simulator to predict the event dates. At each event, various properties of the system can be calculated and accumulated. The timeline is simulated (and relevant parameters accumulated) through many possible time histories using Monte Carlo sampling, to create a statistical interpretation of the life-cycle costs.

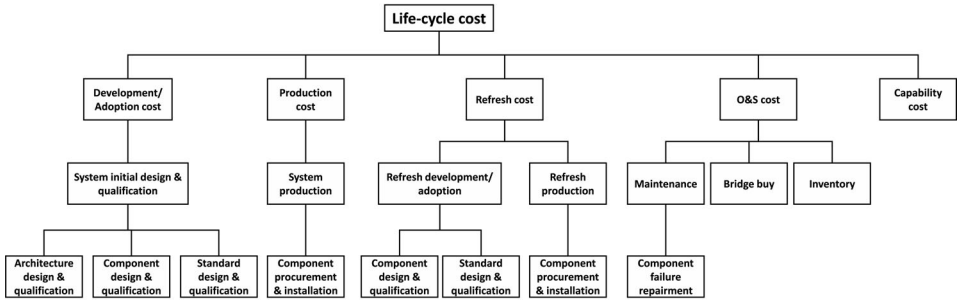
In the discrete-event simulator developed for this paper, the event dates are determined by sampling time-to-failure (TTF) distributions, forecasted component obsolescence date distributions, and a predetermined refresh/redesign schedule. The events of interest are maintenance events, production events (delivery of new systems), retirement events (retirement of fielded systems), logistics events (management of spares, lifetime buys of parts to manage obsolescence) and design redesigns/refreshes.

Sections 2.1 and 2.2 discuss the costs in equation (1) and how they were determined by the cost model. Section 2.2 illustrates how capability cost is valued by model.

2.1 Cost model

Figure 1 shows the structure of the life-cycle cost used in this paper. The life-cycle cost includes five cost categories: development/adoption, production, refresh, operation and support (O&S) and capability. These cost categories were selected because they are sensitive to the degree of system openness. For example, the cost of fuel is one particular cost that may not be affected by openness, so it is not considered in our life-cycle cost structure. Ultimately, we are only interested in the difference in the life-cycle cost between cases with varying degrees of openness, therefore, some costs can be omitted (i.e., they subtract out of the analysis, see Section 3.2.1).

Figure 1 Life-cycle cost structure



2.1.1 Development/adoption costs ($C_{Development}$)

$C_{Development}$ 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 customised components, adopting appropriate COTS components and open standards, as well as the costs of partial or alternative designs considered, but not implemented. Prototyping and design overhead costs are included. $C_{Development}$ also includes the non-recurring engineering (NRE) costs, which may consist of the qualification testing used to demonstrate that the standards, components, subsystems, and the complete system meet performance, reliability, security, and other requirements.

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

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 interactions within the architecture requires higher cost for design and qualification. In addition, a more open architecture means that more open standards are applied at the interfaces in the architecture. Design for open standards requires more activities such as evaluating if current and future components can conform to the standard so that the components can be truly 'plug-and-play'. As a result, the process of design for an open standard may cost more than a proprietary interface.

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 a COTS component results in a smaller design cost than creating a proprietary component since the former can be acquired directly from the market. Each component also requires qualification testing to assure it meets the criteria of performance and reliability. In our model, the total adoption/development cost associated with the components is calculated by summing the design/qualification cost of each type of component that is used in the architecture.

The design and qualification cost of standards 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, our cost model also assumes that the design and qualification cost of standards only depends on the type of standards that are chosen in the architecture.

2.1.2 Production costs ($C_{Production}$)

$C_{Production}$ includes all costs to manufacture and field the system, including component procurement, assembly/manufacturing, stress screening of hardware components (if any), and recurring testing costs.

The production events are generated based on the system's production schedule. In our model, the production (recurring) cost of a system instance is defined as the sum of the costs of the procurement of all the system components and the installation of the components into the final system in the field. The procurement and installation cost of a component depends on the type of component. For the same functionality, the production cost of a COTS component is assumed to be less expensive than a proprietary component.

2.1.3 Operation and support costs ($C_{O\&S}$)

$C_{O\&S}$ are the costs of sustainment incurred from system deployment to the end-of-support, including the costs of: system operation, modifying, maintaining, supplying, training, and inventory supporting. The events associated with sustainment include, but are not limited to: system failure repair, periodic maintenance, sparing management and obsolescence mitigation.

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 (availability impacts are not included, i.e., we assume that they are the same for the cases compared and therefore subtract out – see Section 3.2.1). 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.

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 shelf life of all components is assumed

to be long enough that parts bought in a lifetime buy can be used for the rest of the system support life.

2.1.4 Refresh costs ($C_{Refresh}$)

Through the entire life-cycle, system design refresh may be desirable (or necessary) to assure that the system remains supportable or to maintain the technological capability of the system (see Section 2.2). System refresh replaces the obsolete or technologically out-of-date components with procurable and possibly more technologically advance components. Refresh costs include the cost to develop or adopt the 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 predetermined schedule of refresh development events is provided as an input to the model. The development of a design refresh results in a new baseline architecture, with a new version of components or standards in the architecture. The system follows the current architecture baseline for production and refreshes delivery until the next refresh development occurs.

At a refresh development event, every component and standard in the architecture is examined. If a component is obsolete or is required to be upgraded, it will be refreshed. A refresh-required component might affect other components or standards to be refreshed due to functional dependencies in the architecture.

In our model, a refreshed component is replaced with another component with the same function, same procurement life, and good-as-new reliability. If the refreshed component interfaces with its surrounding components with open standards, a 'plug-and-play' refresh may be achieved, i.e., we could switch the component 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 proprietary link or the open standard is obsolete, the new component may require that the connected components also have to be refreshed.

The structure of refresh development/adoption cost is similar to $C_{Development}$ described in Section 2.1.1 except that the design and qualification cost of the architecture is excluded from the refresh cost. Since the architecture itself does not change throughout the life-cycle, there is no cost associate 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 of the refresh to each system 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 for the particular fielded instance of the system. An example of evaluating the component version in a system instance is demonstrated in the Appendix.

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 Cost of system technological capability ($C_{Capability}$)

Technological superiority is a priority for some critical systems, especially defense systems. With more advanced technological capability, systems are more competitive and the possible threat from adversaries may be reduced. To maintain technological superiority, constant system upgrades may be required. In systems where technological superiority is a priority, the system's upgrade frequency may be fixed and the life-cycle cost is optimised based on this constraint. In other systems, technological capability may be less important and its value can be traded off against other cost factors. This study considers both cases in the case study presented in Section 3:

1 the capability benefit of implementing technology refresh is quantified, as one of the cost factors, i.e., the cost of system technological capability ($C_{Capability}$)

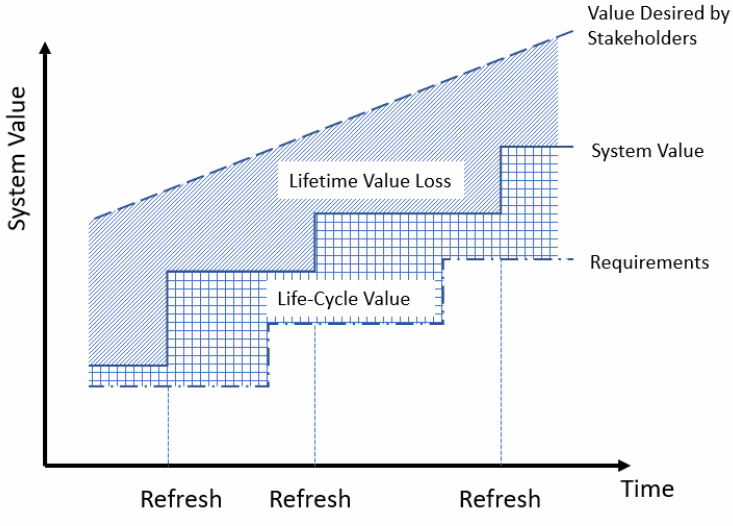
2 open versus closed implementations of a system with a fixed refresh schedule.

In general, capability is defined as "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, n.d.). For the purposes of this paper, we will define the system's technological capability 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 described in Section 2.1, but the 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. Since the effectiveness is strongly tied to system upgrade and refresh, the cost of system technological capability can also be viewed as a metric to quantify the value of a particular refresh plan.

Studies that are related to the concept of capability cost can be found in the area of refresh plan quantification. Figure 2 shows two examples of how a refresh plan can be valued. These studies started by assigning an absolute value to system performance or capability. The system value is a function of time and can be increased by refreshing the system technology. Engel and Browning (2008) assumed that there is an upper limit of system value, called the 'value desired by stakeholders', which is also an increasing function over time, see Figure 2. The lifetime value loss is the area between the actual system value and value desired by stakeholders. Alternatively, Zellers (2016) assumes that the system's minimum requirements over time is a lower bound, and life-cycle value is the area between the two step curves in Figure 2. Both approaches use the area between system value and either the upper bound or lower bound to represent the total life-cycle loss/value (both models are essentially equivalent).

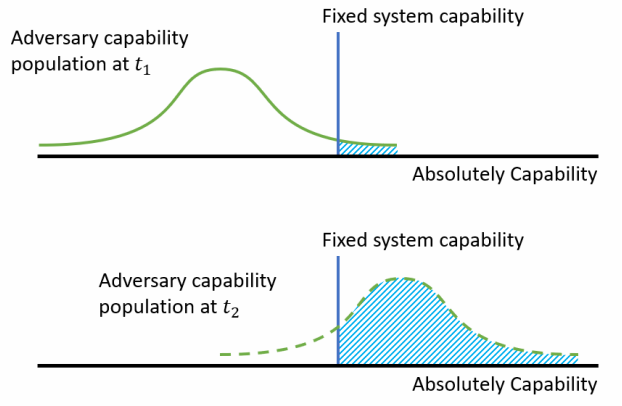
Figure 2 Quantification of the loss/value of a refresh plan (see online version for colours)



In this paper, instead of using an absolute capability metric, e.g., Figure 2, the concept of relative capability is introduced. For a system that is designed to operate in a competitive environment such as defence, 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 3, the distribution curves represent how an adversary's capability evolves over time (from t_1 to t_2), and the vertical line indicates the 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 3 demonstrates that even if a system's absolute capability is fixed (i.e., does not 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 in the effectiveness of the system in performing the mission it was designed to perform, or an increase in the risk of losing the system[4].

The relative capability of a system instance can be represented by the shaded area in Figure 3, the probability that the system capability is less than an adversary's system capability. At a given time point, the probability of the system losing the capability competition can be written as $p(\Delta t)$ where Δt, the technology lag time, represents the technology age difference between the system and the advisory at the time point. That is, at a time point when the system is installed with an older technology, i.e., a positive Δt, the system would have a larger value of $p(\Delta t)$, and therefore is more likely to lose the capability competition to its adversary.

Figure 3 The adversary's absolute capability distribution shifts to the right over time relative to a fixed system capability (see online version for colours)



In the case study presented in Section 3, we assume that the adversary capability follows state-of-practice technology and therefore the Δt is always positive[5]. The Δt of a system at a given time point can be represented as the age of the current architecture used in the system. To be more specific, Δt is the time difference between the current time point and the time of development completion of the architecture currently used in the system, i.e., the time point when the architecture is still state-of-practice. For example, if there is a system instance where its current architecture was first developed in 2002. The technology lag time for this system instance in 2006 is four years. If there is no refresh delivery to the system instance, Δt would only increase over time.

Each delivery of a technology refresh to the fielded system instance resets Δt and the relative capability to a higher level (lower probability of losing the capability competition). Frequent technology refreshes keep the system more up-to-date during its life cycle, reducing the probability of losing the technology competition. Note, although we are discussing the cost of relative capability in the context of a defence application, the concept is relevant to non-defence systems too. For example, a system whose competitiveness in the marketplace depends on constant technology refreshing – the risk could be loss of market share.

To obtain the quantitative cost, we must construct the relationship between relative capability and cost. For a defence system, the cost of system capability may translate into risk, which evaluates the potential loss of systems and missions. The life-cycle risk of a system is the sum of system annual risk over its operation life.

Equation (2) provides the general formulation of the risk cost (cost per unit time) of a system instance in a given time window associated with relative capability.

$$R = IpC_q \tag{2}$$

In equation (2), I represents the expected number of events (encountering an adversary system) in a time window, e.g., 2 times per year. The 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 (measured as a cost per event) if a system loses the capability competition to the adversary.

Based on equation (2), the calculation of the total cost of technological capability is the sum of the annual risk cost of each system instance throughout the life cycle,

$$C_{Capability} = \sum_{i=1}^{N} \sum_{j=1}^{T} Ip\left(\Delta t_{i,j}\right) C_q \tag{3}$$

where N is the number of systems, i.e., fleet size, and T is the total number of support years for the fleet of systems. In equation (3), I and C_q are both assumed to be deterministic values (they could be represented by probability distributions if the appropriate information was available).

For $p(\Delta t_{i,j})$, the relationship between Δt and the probability p was first subjectively determined. The $\Delta t_{i,j}$ of i^{th} system instance (note, system instances are distinguished because not all of the system instances receive the results of a technology refresh at the same time) in j^{th} year can be evaluated based on the production/refresh schedule and the adversary technology evolution assessment process. The corresponding p can be determined using the Δt in the subjective function $p(\Delta t)$. The product $Ip(\Delta t_{i,j})C_q$ models the expected annual cost of the system instance i in year j, given the discrete technology lag time Δt.

3 A-RCI case study

In this section, we present a case study of the acoustic rapid COTS insertion (A-RCI) sonar system. The A-RCI program, implemented a COTS-based open architecture for a submarine sonar signal processing system. The A-RCI eliminated traditional system architecture that used specialised proprietary components that were built to military specification. Embracing the use of COTS and commercial standards, allowed for the sonar signal processing system to be upgraded, without altering other sonar system components (Guertin and Miller, 1998). In addition, the A-RCI case is also regarded as a successful example of open-systems architecture. One study cites preliminary results compiled from ten years of data on both the acoustic rapid COTS insertion program and its predecessor indicating a life-cycle cost improvement of nearly 5:1 (Boudreau, 2006).

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 phases 1 and 2, A-RCI developed a multi-purpose processor (MPP) to process the data from both a towed array (TA) and a hull array (HA). Phase 3 added spherical array MPP and switch MPP to replace the legacy system spherical array processing functions. Phase 4 integrated another high-frequency sail array MPP into A-RCI. By the end of phase 4, a COTS-based open-architecture A-RCI system completely replaced the original legacy system.

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

We wish to clarify that the data describing the A-RCI in this section represents our interpretation of the A-RCI and the A-RCI program, and may not exactly match the actual system or program. The A-RCI is a defence system that spans many years, and as such, a complete dataset describing the A-RCI is not publicly available.

Table 1 Input parameters for modelling the A-RCI

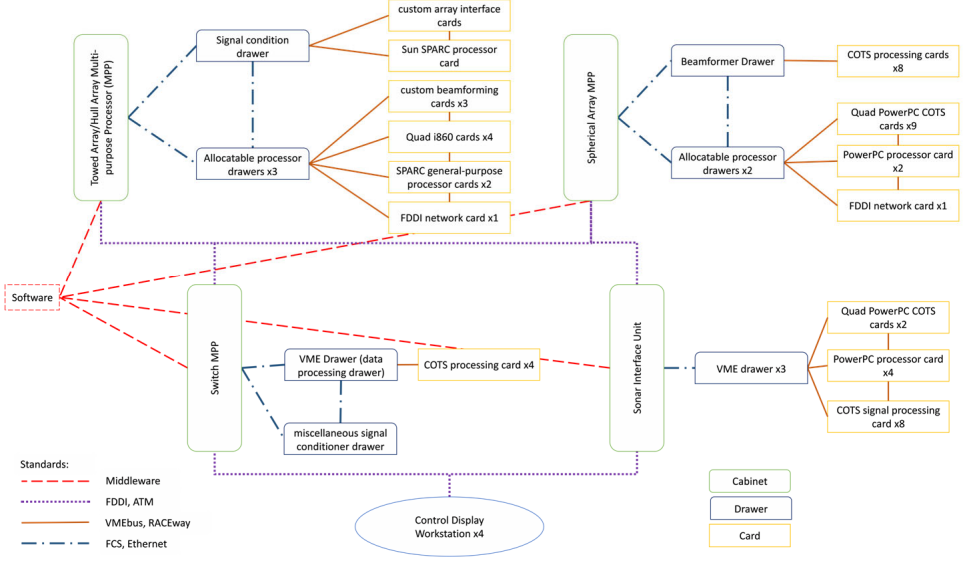
Input parameters		*Input value*	*Source*
Architecture		Figure 4	Guertin and Miller (1998)
Production schedule		Figure 6	Schuster (2007)
Retirement schedule		Figure 6	Schuster (2007)
Architecture R&D cost	Phase I	$48,350,000	From calibration, see Section 3.1.1
	Phase II	$39,015,000	
	Phase III	$55,745,000	
	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

3.1 A-RCI input data

In this section, the data describing the A-RCI case is provided. Table 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.

The architectural input data was based on Guertin and Miller (1998). Figure 4 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 connections between the cabinets are indicated as middleware. Open standards are also represented by the linkages shown in Figure 4. The actual number of components (cards) is larger than those shown in Figure 4. The components that are common to all system architectures, i.e., they would be the same whether an open or closed system are 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 3.2.1).

Figure 4 Architecture assumed in the A-RCI case study (see online version for colours)

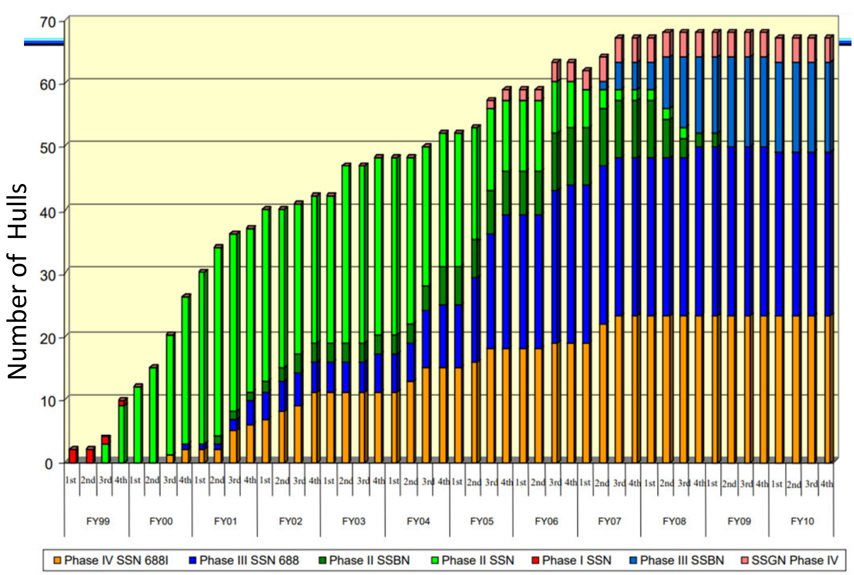


Due to the complexity of the A-RCI architectures, we adopted a design structure matrix (Eppinger and Browning, 2016) approach to model the A-RCI. The design structure matrix captures how the system's components interact with each other. Capturing the component interactions is necessary to cost the design refreshes, i.e., when a particular component is changed at a refresh, other required component changes are captured by the design matrix; similarly, the interactions captured in the design matrix guide re-qualification requirements for the system at refreshes.

Figure 5 Partial design matrix of the A-RCI phase 3 architecture

	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 6 Installation profile for the A-RCI (see online version for colours)



Source: Schuster (2007)

Because of space constraints, Figure 5 only shows the portion of the design structure matrix of the A-RCI phase 3 architecture that represents the architecture of TA/HA MPP. For an element a_{ij} in the i^{th} 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 dependency between components i and j. Different values of a_{ij} represent different types of interfaces between components i and j. The values of a_{ij} have no quantitative meaning, they just enumerate distinct connections: positive integers represent the open standards used in A-RCI (1: middleware; 2: FDDI, ATM; 3: VMEbus, RACEway; 4: FCS, Ethernet). The modules shown in Figure 5 are adopting all open standards.

The installation and retirement profile and number of system instances of the A-RCI is shown in Figure 6 (Schuster, 2007).

3.1.1 Data calibration

The components and the standards 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. To determine these values, we reverse-engineered 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 summarised 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 the architecture 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, we first estimated the refresh development cost per year intermediate variable, based on the total development cost from ASSET and the refresh schedule used for the A-RCI. The average refresh cost per component was the refresh development cost per year divided by the number of refreshed components in each.

The details of calibration process can be found in (Chen, 2022). The results of the calibration process are included in Table 1.

3.2 Modelling results

In this study, two system configurations were compared based on their life-cycle cost:

Original A-RCI	the actual A-RCI architecture, more open.
Closed version	a less-open version of A-RCI where two of the modules (Spherical Array MPP and Switch MPP in Figure 4) adopted closed standards and proprietary components were used instead of COTS parts.

Both configurations followed the same production/retirement schedule (Figure 6) and used the same input data described in Table 1.

In sections that follow, the concept of a relative life-cycle cost is introduced using an example simulation result. Selected sensitivity analysis was performed to evaluate how three main key parameters, risk profile, end-of-support year and fleet size, affect the simulation result.

3.2.1 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. To assess the cost of openness we are actually only interested in the difference in the costs discussed in Sections 2.1 and 2.2 between the two system implementations. 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) do not have to be modelled because they subtract out. The cost difference between two cases is significantly easier to determine than the absolute cost of the cases. The model described in this paper never produces absolute costs, only the cost differences between two cases.

Figure 7 The relative cost model approach to obtain the accumulated cost difference (expected consequence per capability competition loss is C_q = \$1 million, end-of-support = 36 years, 6-year refresh interval) year 0 is FY99 (see online version for colours)

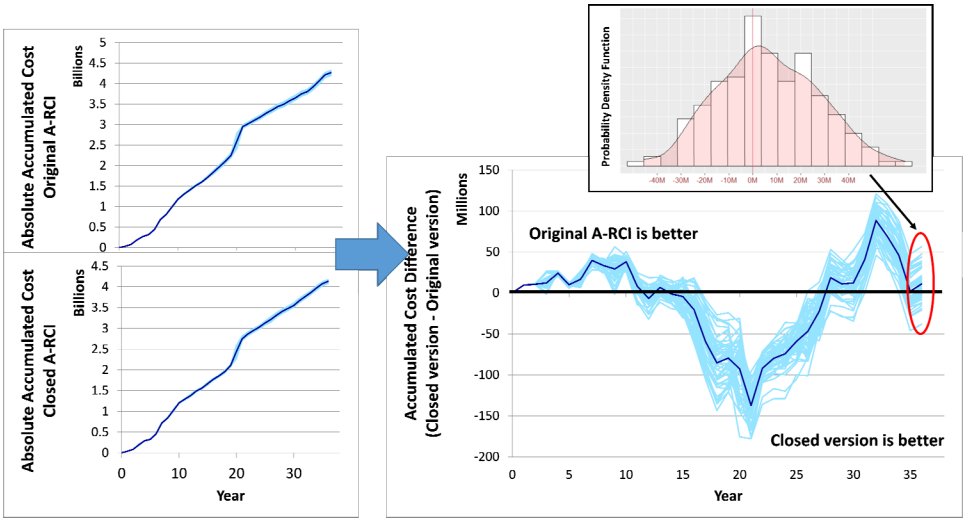


Figure 7 shows the process of obtaining relative cost, including an example result from the simulation model. On the left side of Figure 7 are the absolute accumulated costs for the entire fleet of two scenarios calculated from the discrete-event simulation. In both cases many time-history solutions are used (each is the result of a unique combination of samples from the input probability distributions, so each is one possible time history for the fleet). The darker solid line in the right figure is the mean of the time histories. The absolute accumulated costs for the two system configurations on the left side of Figure 7 are NOT particularly meaningful, because the relevant life-cycle costs that are not affected by system openness are omitted. The right side of Figure 7 shows the difference between the two system costs. The vertical axis in the right figure is the life-cycle cost difference. Figure 7 shows that at the end of 36 years, the mean value of the accumulated

cost different is greater than zero, indicating that the original version of the A-RCI will have a lower life-cycle cost than the closed version for this example case.

Figure 8 Life-cycle cost difference as a function of end-of-support year (expected consequence per capability competition loss is C_q = \$1 million, 6-year refresh interval) year 0 is FY99 (see online version for colours)

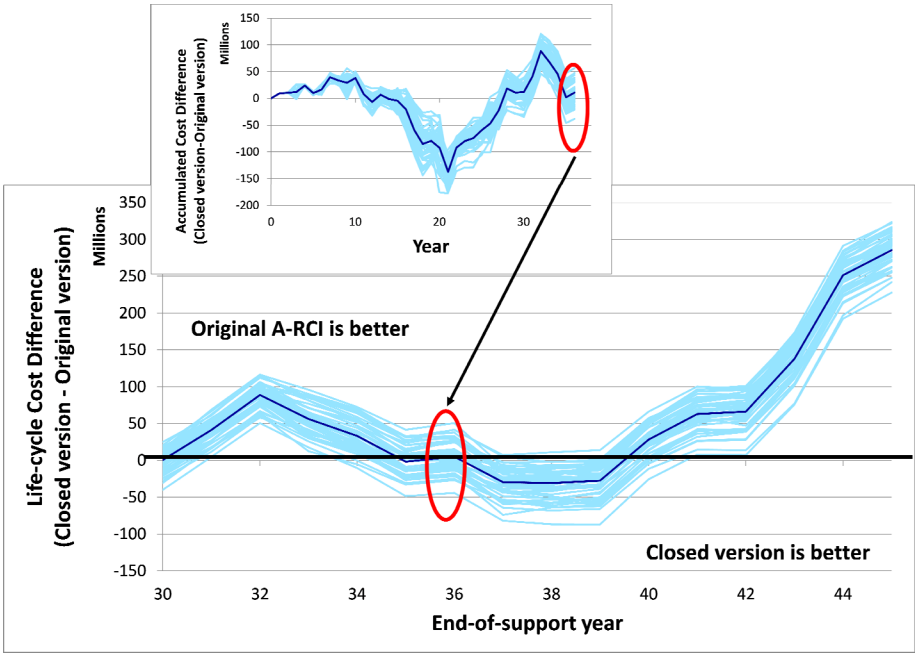


Figure 8 shows the life-cycle cost difference as a function of end-of-support year.[6] The result in Figure 7 is the 36 year end-of-support data point in Figure 8. In Figure 8, a positive result indicates that given the end-of-support life, the original A-RCI configuration is more cost effective, while a negative result value indicates that the less-open version of A-RCI is more cost effective. This result can be interpreted as: for the assumed risk profile, if the end-of-support year is between 35 to 39, we should choose the less-open version of A-RCI rather than the original A-RCI. The results shown in Figures 7 and 8 represent one specific case to demonstrate the methodology. In the sections that follow we will conduct a more detailed analysis comparing the open and closed versions of the A-RCI.

3.2.2 The effect of refresh strategy

In the previous section, we demonstrated how the simulation can be used to assess the life-cycle cost difference between two system configurations. In this section, we examine how the best refresh strategy could be determine based on its effect on the life-cycle cost.

Refresh strategy affects both refresh cost and capability cost. More frequent refreshing costs more, but, a more frequent refresh strategy 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.

We first compared the life-cycle cost of the original (open) A-RCI and the closed version assuming the same A-RCI refresh cycle, i.e., two-year refresh interval. The result in Figure 9 shows that the original A-RCI is always more beneficial than the closed version (for a 2-year refresh interval).

We next varied the refresh strategy of the less open version of A-RCI, 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 intervals shown in Figure 9. In this case, no-refresh is the optimum refresh strategy for the closed version of A-RCI since the curve is below the other cases and below 0 in Figure 9. The closed version of A-RCI is more cost effective than the original as the no-refresh curve is negative in the end-of-support year range from 30 to 45.

Figure 9 Life-cycle cost difference given different refresh strategies (C_q = \$1M assumed) year 0 is equivalent to FY99 (see online version for colours)

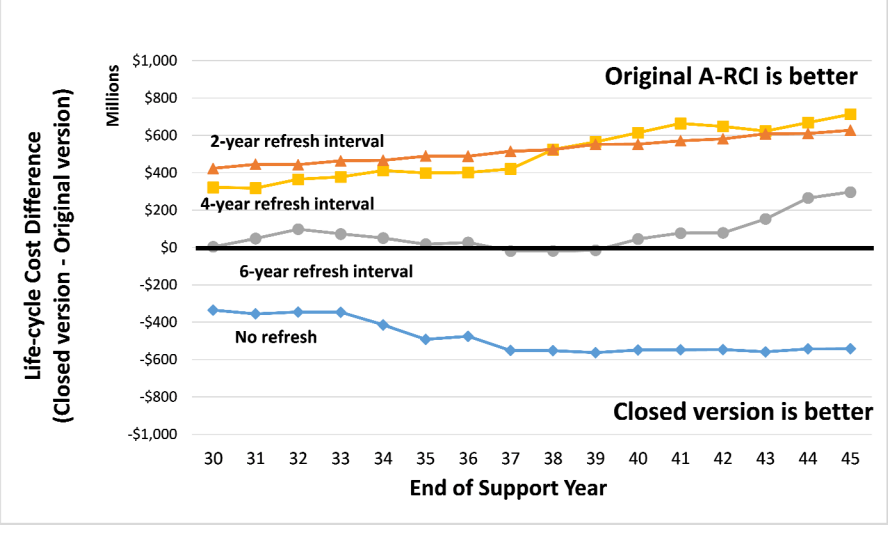


Figure 9 indicates that more frequent refresh increases the life-cycle cost difference between the closed version and the original version. As stated previously, refresh cost and capability cost are two cost factors that are directly influenced by the frequency of refresh. Frequent refresh results in high refresh cost but lower capability cost. Figure 9 shows that for a \$1M consequence cost, the increase in refresh cost is more significant than the decrease in the capability cost penalty.

3.2.3 Sensitivity analysis

In addition to the length of life cycle (end of support year) and the refresh strategy, in this section, we further analysed how other external factors 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 characterised as the consequence of losing the capability competition to adversary systems and the frequency of encountering an adversary system, i.e., the product of I and C_q in equation (2). 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 10 Cost difference comparison of 30-year support life given different risk profiles and refresh strategies (see online version for colours)

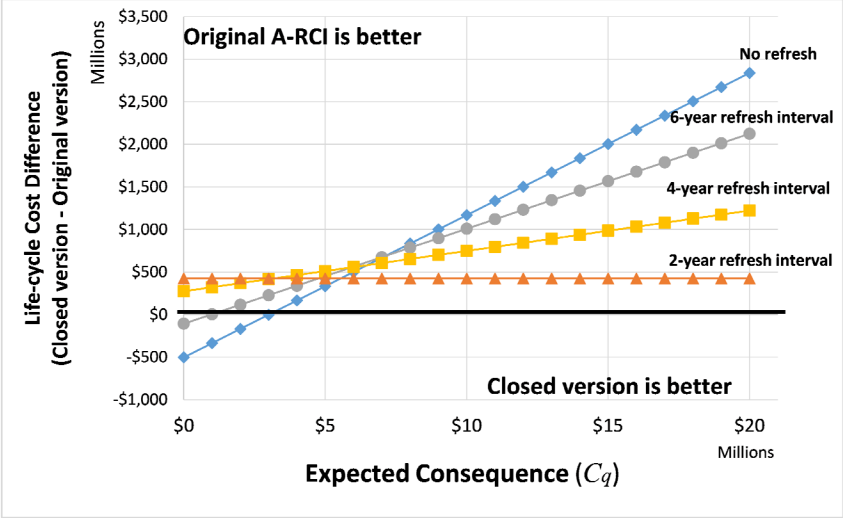


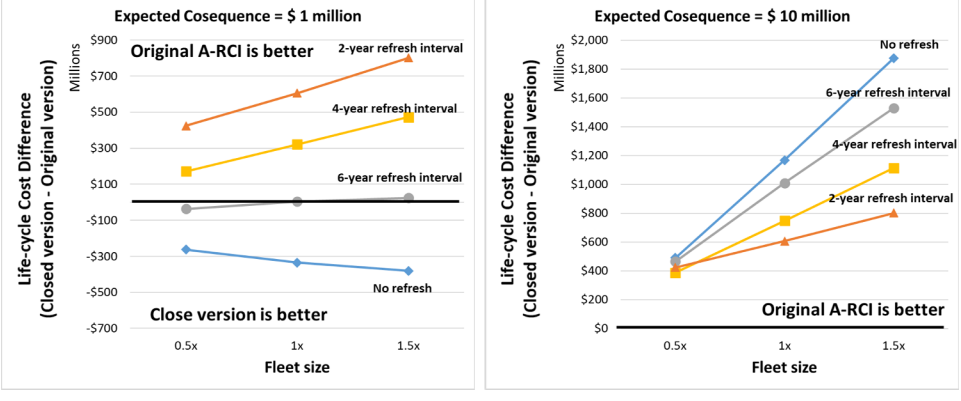
Figure 10 shows the cost difference between the two configurations (the original A-RCI with its 2-year refresh and the closed architecture A-RCI with various refresh intervals) all of which have a 30-year support life. Each data point is the mean of one simulation cost difference result. The four curves shown represent different refresh strategies that were implemented to the closed configuration. In Figure 10, the data points that have a positive cost difference are the results for which the original A-RCI configuration is more cost effective, 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 10. Since the two configurations both follow the same A-RCI refresh cycle, theoretically, the technology performance would be the same throughout the life cycle in both configurations. Therefore, there would be no sensitivity to the capability cost.

Based on Figure 10, we can observe how the risk profile and the refresh strategy jointly affect the result. First, the three curves in Figure 10, 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 increasing in capability cost of the original A-RCI is less than the increasing in capability cost in the closed version of the 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 (C_q) is less than \$6 million, no refreshes represents 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 cost effective 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 favourable.

Figure 11 Cost difference comparison of 30-year support life given different fleet sizes and refresh strategies (see online version for colours)



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 11 shows two sensitivity analyses of the number of fielded systems under low and high-risk profiles separately. In Figure 11, 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 size. As the fleet size grows, the influence of the fleet-size-dependent cost factors has 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 cost effective or not. On the left side of Figure 11, 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 cost effective 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 cost effective over the fleet size. On the right side of Figure 11, 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.

3.2.4 A-RCI refresh plan as a mandate

In Section 2.2, we mentioned that there are cases where the required system capability and performance over time is not traded off against life-cycle cost. In these cases, the

refresh schedule is predetermined and treated as a constraint in the decision-making process. In these cases, the cost difference between the open and closed versions of ignores the capability cost ($C_{Capability}$) since both versions adopt the same refresh plan. We considered these refresh-fixed cases by assigning both open and closed versions the A-RCI a 2-year refresh plan. The results for this case are the orange triangle lines in Figure 9 through Figure 11. In all situations, the original open A-RCI always has a lower life-cycle cost than the closed version. The results can be interpreted as, with the 2-year refresh constraint, the original A-RCI architecture is a more cost effective way to sustain the A-RCI systems.

4 Discussion and conclusions

The goals of implementing an open system architecture (OSA) are often defined in qualitative terms. One clear benefit is that an OSA can reduce the potential for 'vendor lock', i.e., it can increase the buyers' power over vendors by enabling multi-vendor competition during virtually all phases of a system's life cycle. This increased competition can also incentivise improved vendor performance and innovation, driving prices lower at the same time (Guertin and Womble, 2012). 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 paper presents a framework for the quantitative analysis of OSA by modelling the life-cycle cost difference associated with system openness, including open architecture, open standard and commercial off-the-shelf (COTS). The cost impact of openness is evaluated by including these concepts of openness into lifetime events and corresponding costs.

A case study of the A-RCI has been used to demonstrate the application of quantitative analysis on cost in relation to system openness. The life-cycle cost difference between the original A-RCI and a less open version of A-RCI was evaluated. Sensitivity analysis was performed to determine how the important factors, including refresh strategy, end-of-support year, risk profile and fleet size, affected the comparison result. Given the condition that the expected consequence of capability loss (C_q) is less than \$3 million, the closed version was found to be more cost effective than the original A-RCI architecture and its 2-year refresh plan, when the closed version did not refresh. Given a fixed 2-year refresh plan for both versions, the original A-RCI architecture is always better. It should be noted that the results presented in this paper are highly dependent on the input data that consists of our interpretation of the A-RCI and the A-RCI program, which may not be representative of the actual system or program. It should also be noted that programs like the A-RCI are not composed of just hardware and software, but also people whose behaviour, experience, and training all contribute to the final life-cycle cost of the system.

Future work will include several elements. More details including shelf life impacts of components purchased in lifetime buys and other types of inventory management realities could be included. More openness impacts could also be considered in the model. For example, examining the impact of developing open-source software. It is believed that open source reduces the development cost since a wider community of developers are able to review and test the code. However, when the source code is published openly, hackers can easily find and exploit vulnerabilities in the software.[7]

Thus, the trade-off is that more money would need to be spent on cybersecurity enhancement. Second, further study will focus on developing the relationship between system openness and life-cycle cost. For example, a model of life-cycle cost associated with COTS functional density has been proposed from a software perspective (Abts, 2002). For a complex system integrating both hardware and software, a more sophisticated model is needed. Moreover, since system openness is not just about the number of COTS, a quantitative metric for system openness should also be developed.

Acknowledgements

Funding for this work is provided by Naval Postgraduate School (Grant Number HQ00341910006). We would also like to thank ASSETT, Inc. and Mr. William Johnson for providing us with valuable insights on the cost of the A-RCI system.

References

ASSETT (2006) *Submarine Sonar Cost Analysis Report*.

Abbott, J.W., Levine, A. and Vasilakos, J. (2008) 'Modular/open systems to support ship acquisition strategies', *Proc. Am. Soc. of Naval Engineers Day*, Arlington, VA.

Abts, C.M. and Boehm, B.W. (1997) *COTS Software Integration Cost Modeling Study*, Contract, 30602(94-C), 1095.

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

Abts, C. (2002) 'COTS based systems (CBS) functional density – a heuristic for better CBS design', *Proceedings of the First International Conference on COTS-Based Software Systems*, Springer, Orlando, FL, pp.1–9.

Boeing (n.d.) *KC-46A Pegasus* [online] https://www.boeing.com/defense/kc-46a-pegasus-tanker/ (accessed January 2022).

Boudreau, M. (2006) Acoustic Rapid COTS Insertion: A Case Study in Spiral Development. Naval Postgraduate School Report.

Chen, S-P. (2022) 'Appendix A: Calibration of the A-RCI case study', *Analysis of the Life-Cycle Cost and Capability Tradeoffs Associated with the Procurement and Sustainment of Open Systems*, PhD dissertation in the Department of Mechanical Engineering, University of Maryland.

Clark, B. and Clark, B. (2007) 'Added source of costs in maintaining COTS-intensive systems', *Cross Talk, The J. of Defense Software Engineering*, Vol. 20, No. 6, pp.4–8.

Clough, A. (2003) *Commercial-off-the-Shelf (COTS) Hardware and Software for Train Control Applications: System Safety Considerations*, No. DOT-VNTSC-FRA-02-01, John A. Volpe National Transportation Systems Center (US).

DoD (2015) *DoD Instruction 5000.02, Operation of the Defense Acquisition System* [online] https://www.acq.osd.mil/fo/docs/500002p.pdf (accessed 22 December 2020).

DoDAF (n.d.) *DM2 Data Groups* [online] https://dodcio.defense.gov/Library/DoD-Architecture-Framework/dodaf20_capability_mm/#:~:text=A%20capability%2C%20as%20defined%20here,published%20by%20the%20Joint%20Staff (accessed January 2022).

Dunn, S., Laroche, J. and Mitchell, P. (2018) 'Open system architectures for the ADF: opportunities and challenges', *Australian Defence Force Journal*, No. 204.

Engel, A. and Browning, T.R. (2008) 'Designing systems for adaptability by means of architecture options', *Systems Engineering*, Vol. 11, No. 2, pp.125–146.

Eppinger, S.D. and Browning, T.R. (2016) *Design Structure Matrix Methods and Applications*, The MIT Press, Cambridge, MA.

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

GAO (2014) *Review of Private Industry and Department of Defense Open Systems Experiences*, GAO-14-617R.

Guertin, N.H. and Miller, R.W. (1998) 'A-RCI – the right way to submarine superiority', *Naval Engineers Journal*, Vol. 110, No. 2, pp.21–33.

Guertin, N.H. and Womble, B. (2012) 'Competition and the DoD marketplace', *Proceedings of the Ninth Annual Acquisition Research Symposium*, NPS-AM-12-C9P17R01-076.

Grant, J., Rankine, R., Brown, K.M., Carter, W.R. and Foreman, J. (2000) *Ensuring Successful Implementation of Commercial Items in Air Force Systems*, Scientific Advisory Board (Air Force), Washington DC.

Hanratty, J.M., Lightsey, R.H. and 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) *The Case for Open Systems Architecture*, 14 December [online] http://pmh-systems.co.uk/Papers/MOSACaseFor/ (accessed January 2022).

Hoepman, J-H. and Jacobs, B. (2008) 'Increased security through open source', *Communications of the ACM*, Vol. 50, No. 1, pp.79–83.

Jensen, F. and Petersen, N.E. (1982) *Burn-in: An Engineering Approach to the Design and Analysis of Burn-in Procedures*, Wiley, New York.

Lewis, P., Hyle, P., Parrington, M., Clark, E., Boehm, B., Abts, C. and Manners, R. (2000) *Lessons Learned in Developing Commercial Off-The-Shelf (COTS) Intensive Software Systems*, Federal Aviation Administration Software Eng. Resource Center Report.

Logan, G.T. (2004) *The Modular Open Systems Approach (MOSA)*, OSJTF Presentation to the Exec. Prog. Managers Course [online] http://acc.dau.mil/CommunityBrowser.aspx?id=37585.

Mladenović, D., Jovanović, D. and Denić, N. (2013) 'Open source solutions in the development of military unmanned aerial systems', *Scientific Technical Review*, Vol. 63, No. 1, pp.36–46.

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.

OSJTF (2004) *Open Systems Joint Task Force, 'Program Manager's Guide: A Modular Open Systems Approach (MOSA) to Acquisition'*, US Department of Defense [online] http://www.acq.osd.mil/osjtf/pdf/PMG_04.pdf (accessed September 2004).

Peruzzi, L. (2019) 'A new generation of submarine combat management systems', *European Security & Defence* [online] https://euro-sd.com/2019/05/articles/13130/a-new-generation-of-submarine-combat-management-systems/ (accessed 29 December 2020).

SAF Public Affairs (2020) *Air Force, Boeing Agree on Final KC-46 RVS 2.0 Design*, 2 April [onlne] https://www.afrc.af.mil/News/Article/2135364/air-force-boeing-agree-on-final-kc-46-rvs-20-design/ (accessed January 2022).

Sandborn, P. (2017) *Cost Analysis of Electronic Systems*, 2nd ed., World Scientific, Singapore.

Schramm, Z. (2013) *A Model for Estimating the Cost Tradeoffs Associated with Open Electronic Systems*, MS thesis, Dept. of Mechanical Engineering, University of Maryland.

Schuster, J. (2007) *Recent Progress in Submarine Sonar … and Future Challenges in ASW*, MI Department of Mechanical Engineering, Center for Ocean Engineering, Ocean Acoustics Group 7 [online] http://acoustics.mit.edu/dyerparty/Ira%20Dyer%20talk%20rev%201.pdf (accessed January 2022).

Taylor, D. (2018) 'Why Linux is better than Windows or macOS for security', *Computerworld*, 6 February 2018 [online] https://www.computerworld.com/article/3252823/why-linux-is-better-than-windows-or-macos-for-security.html (accessed January 2022).

UK MoD (2013) *Guidance System of Systems Approach (SOSA) Open Systems Strategy* [online] https://www.gov.uk/government/publications/system-of-systems-approach-sosa-open-systems-strategy (accessed 23 December 2020).

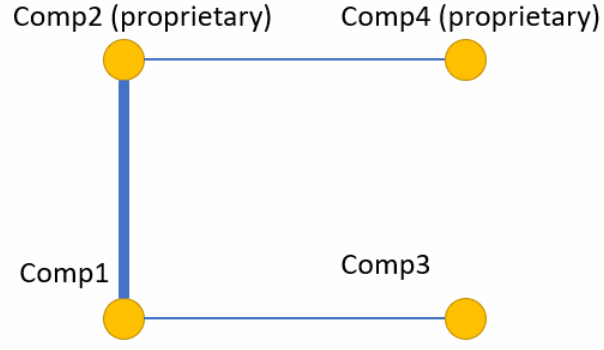
Wright, M., Humphrey, D. and McCluskey, P. (1997) 'Uprating electronic components for use outside their temperature specification limits', *IEEE Transactions on Components, Packaging, and Manufacturing Technology, Part A*, Vol. 20, No. 2, pp.252–256.

Zellers, E.M. (2016) *Design of Flexible Technology Refresh Plans for Military Open Systems Architectures*, PhD dissertation in the Department of Aerospace Engineering, Georgia Institute of Technology.

Appendix

This appendix presents a simple example to demonstrate the life-cycle cost calculation used in this paper. For demonstration purposes this example is deterministic, however, in the A-RCI case presented in Section 3 of the paper, many of the inputs are probability distributions. The example architecture considered is a four-component/three-interface system shown in Figure A1. Components 1 and 3 are COTS. Components 2 and 4 are proprietary. The interfaces of components 2, 4 and components 1, 3 are open standards and the interface of component 1, 2 is a proprietary interface. Table A1 shows the parameters used in this example.

Figure A1 The example architecture (see online version for colours)



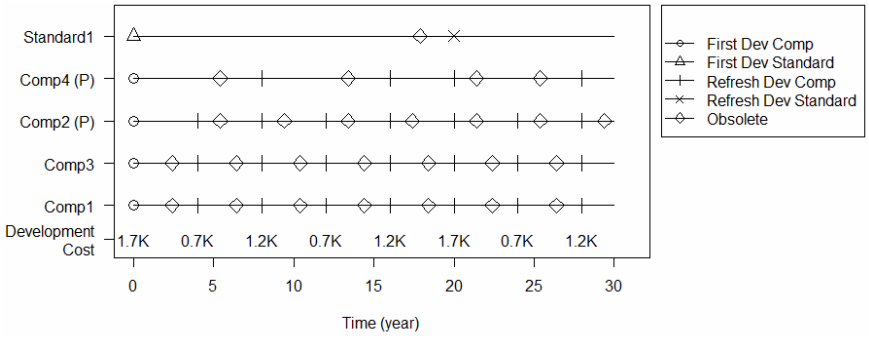
Note: The thick line denotes a proprietary interface; the thin line denotes an open standard.

Table A1 Example analysis parameters

	Reliability (year)	*Support life (year)*	*Design cost*	*Procurement cost*	*Install cost*
Proprietary	W(1.75,8)	5.4	$500	$2	$0.2
COTS	W(1.75,8)	2.4	$100	$1	$0.1
	Support life (year)		*Design cost*		
Open standard	18		$500		
System	*1*	*2*	*3*	*4*	
Production time (year)	0	1	2	3	
Retire time (year)	30	30	30	30	
Holding cost	$1/year per part				
Discount rate	0				

Step 1 Establish the refresh schedule[8] for the architectural baseline that is based on the obsolescence dates of the components and the standard. In Figure A2, at the first refresh ($t = 4$), components 1 and 3 are refreshed because they are obsolete. Component 2 is also refreshed since it is connected to component 1 with proprietary interface. At the second refresh ($t = 8$) all components are refreshed since they are all obsolete. At the 5th refresh ($t = 20$), since the open standard is also obsolete, all the components and the open standard are refreshed. Once the required components at each design refresh are determined, the corresponding refresh development cost is the sum of the design cost of the refreshed components/standard.

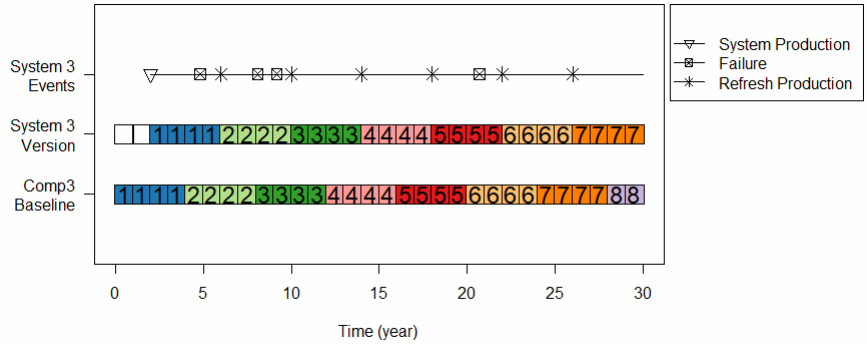
Figure A2 The architectural baseline timeline



Step 2 Generate the system-level production and failure events. In Figure A3, we use component 3 and system 3 as an example. We first build the baseline version and the system version. The component 3 baseline version is based on the 'Comp3' timeline in Figure A2. The version increases by one when there is a refresh development event occurring. System 3 is first produced at $t = 2$ and has a refresh interval of four years. Thereafter, every four years of system 3 production ($t = 6$, 10, 14 and so on), the component 3 on system 3 is upgraded to the version based on the version in the baseline at the same year. In this case, there is a system production event at $t = 2$. The refresh production events are at

the times when the system component receives a refresh installation (version change). For the failure events, starting at the initial production, time-to-failures are sampled and put on the timeline until the next refresh production. This process should be done for all components in all systems.

Figure A3 The component 3 and system timelines (see online version for colours)



Note: The number in each block (year) represents the version

Step 3 Make the required component bridge buy[9] to support all post-obsolescence events. Once all system events are generated in step 2, for the same component, combine and sort all system-level events and the obsolescence events based on version and time. As shown in Table A2, The value of stock at component obsolescence, representing the number for the bridge buy, is the same as the number post-obsolescence events with the same version. At each of the post-obsolescence events, the stock number decreases by one as a component is taken out of the inventory.

Step 4 Evaluate the event cost. At the 'obsolete' event, the cost is the procurement cost of the bridge buy. For the events 'system production', 'refresh production' and 'failure', if the component is not obsolete yet, the event cost is the component procurement cost plus the installation cost. If these events are after the obsolete date, the cost becomes the installation cost plus the inventory cost. The inventory cost is the multiplication of the stock number, holding time and the holding base cost.

Table A2 Partial events of component 3 and the corresponding event costs

Time	*Type*	*Version*	*System*	*Stock*	*Holding time*	*Inventory cost*	*Bridge buy cost*	*Procurement + installation cost*
0.00	System production	1	1	0	0.00	0.00	0	1.1
1.00	System production	1	2	0	0.00	0.00	0	1.1
2.00	System production	1	3	0	0.00	0.00	0	1.1
2.40	Obsolete	1	0	3	0.00	0.00	3	0

Table A2 Partial events of component 3 and the corresponding event costs (continued)

Time	*Type*	*Version*	*System*	*Stock*	*Holding time*	*Inventory cost*	*Bridge buy cost*	*Procurement + installation cost*
3.00	System production	1	4	3	0.60	1.80	0	0.1
3.30	Failure	1	2	2	0.30	0.61	0	0.1
4.80	Failure	1	3	1	2.69	2.69	0	0.1
4.00	Refresh production	2	1	0	0.00	0.00	0	1.1
5.00	Refresh production	2	2	0	0.00	0.00	0	1.1
6.00	Refresh production	2	3	0	0.00	0.00	0	1.1
6.40	Obsolete	2	0	4	0.00	0.00	4	0
6.92	Failure	2	2	4	0.52	2.09	0	0.1
7.00	Refresh production	2	4	3	0.08	0.24	0	0.1
8.09	Failure	2	3	2	1.09	2.19	0	0.1
9.18	Failure	2	3	1	1.08	1.08	0	0.1

Step 5 Repeat Step 1 to Step 4 for sufficient trials (sampling the time-to-failure distributions to generate demands) to generate a total life-cycle cost distribution.

Notes

1 This paper uses OSA to refer to Open Systems Approach in general and MOSA to refer to the US DoD approach specifically.

2 The KC-46A is a widebody, multirole aircraft that can refuel military aircraft inflight, and can also carry passengers, cargo, and patients (Boeing, n.d.).

3 Some preliminary work done formulating a model based on the equation (1) appears in (Schramm, 2013).

4 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 their relative capability of the system decreases over time.

5 This doesn't mean that all adversary systems are necessarily state-of-practice, but that the capability of the adversary population has the same trend as the state-of-practice.

6 The accumulated cost difference in each year in Figure 7 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.

7 Consequently, it is believed that proprietary software is better protected against external attacks. Experience shows that when open-source code is actively reviewed, it has proven to be secure (e.g., the popular operating system Linux) (Taylor, 2018). However, this may not be the case with software that has a limited distribution. This may also not be the case with air-gapped systems like the A-RCI. Some also argue the vulnerabilities are more quickly detected in open systems (Hoepman and Jacobs, 2008).

8 At each design refresh, obsolete or upgrade-required components are refreshed. In addition, these components might cause other components that are connected to them with proprietary interfaces to be refreshed.

9 When the component becomes obsolete, a sufficient number of components are purchased and held in the inventory to support the fleet until the next refresh.