Process of Measuring the Maintainability of Commercial Off-the-Shelf (COTS) Based Systems: A Complexity Approach

***Smith M.*[1], *Lawson D. W.* [2], *Ertas, A.*[3], *and Surles, J.*[4]**

[1] Raytheon Company, Richardson, Texas, USA, mwsmith@pobox.com

[2] Texas Tech University, Civil Engineering Department, Lubbock, Texas, 79409, USA, william.d.lawson@ttu.edu

[3] Texas Tech University, Lubbock, Mechanical Engineering Department, Texas, 79409, USA, atila.ertas@ttu.edu

[4] Texas Tech University, Department of Mathematics & Statistics, Lubbock, Texas, 79409, USA, james.surles@ttu.edu

* **Correspondence:** mwsmith@pobox.com

Received 2 January, 2020; Revised 21 February, 2020 Accepted 25 February, 2020

Available online 28 February, 2020 at www.atlas-journal.org, doi: 10.22545/2020/0132

This paper defines a process of predictive approach to evaluate the maintainability of a Commercial Off-the-Shelf (COTS)-based system (CBS) by analyzing the complexity of the deployment of the system. The approach integrates architectural dependencies and the system's concept of operations to derive a network-based representation of the software system. A greater understanding of the deployment complexity is gained by using a Design Structure Matrix (DSM) to determine the number of architectural dependencies on a COTS product, or in-degree, for each COTS product in the system. The arithmetic mean of the in-degree for all nodes in the system is then compared with the perceived effort to maintain the system. The resultant measure – the in-degree mean – is useful in evaluating the maintainability of the operational system while the system is being designed and throughout its lifetime. Architects can use the approach to assist in COTS product selection and to make product trades to optimize the maintainability of the system. Integrators can use the approach to optimize product deployment and to determine the upgrade strategy for deployment. Finally, maintenance engineers can use the approach to estimate the effort required to maintain the system and to identify areas in which extensive product expertise is required. Because the approach requires only basic information about the system, it can be applied early in the design process and used until the system is decommissioned.

Keywords: COTS, complexity measures, system design, system maintenance, system integration.

1 Introduction

This paper describes a process to evaluate the maintainability of Commercial Off-the-Shelf (COTS) based systems (CBS) during the design and operational phases of a software development project. The approach integrates complexity theory, graph theory, Design Structure Matrix (DSM) theory, network theory, and systems engineering to derive a single predictive measure of the maintainability of a CBS, the in-degree mean. The approach enables all disciplines involved in the development of a CBS to work from a single maintainability metric to deliver a cost-effective solution.

Little evidence exists to suggest that current prediction techniques for software maintainability are effective [1,2] for CBS. The effort associated with maintaining CBS is not generally part of the development and costing models that are popular today; therefore, the effort to deploy and integrate these products is rarely bid or scheduled appropriately. Instead, most existing models are focused around the effort for software engineers to maintain glue-code or custom code, i.e., the non-commercially derived components of a CBS. Even when costing and scheduling models account for COTS integration, they are generally focused on the front-end of the development cycle where the effort is expended on fulfilling mission-specific requirements. Therefore, a useful measure of maintainability is required for CBS that encompasses effort in all phases of development.

While this problem may seem trivial, particularly in the modern design paradigm where a single machine instance is used for a single COTS product, the complexities driven by the dependencies between COTS products increase costs in the system development and deployment timeframes and eventually in the maintenance timeframe. The costs in the maintenance phase of the system are often the most significant since the engineers who are most knowledgeable about the dependencies and interactions of the products are no longer available because they have moved on to other development efforts or because the maintenance team is a completely separate team without reach-back to the development team. Additionally, a traditional divide exists between the operations and development engineers, and the relationship is often antagonistic [3]; therefore, a cultural opportunity also exists to enable the two groups to work more closely and, together, develop a more cost-effective solution. For example, the recently popular DevOps, or Development Operations, system delivery paradigm attempts to accomplish this pairing with software engineers developing and delivering custom software solutions in a rapid-fire series of small incremental deliveries [4].

Existing measures of the maintainability of CBS are focused on the number of COTS products that are part of a system [5,6]. However, as with biological systems, it is not only the number of components that create complexity, but instead the interactions between the components [7,8]. The measure of maintainability described herein is a metric that is based on the interactions of the components.

In this work, the in-degree mean model - a measure of the effort required to maintain the COTS installation - is described as well as the steps necessary to create a network-based model of the CBS. The model is compared to the perceived effort required to maintain 13 operational CBS. The advantages of the approach are discussed along with who can benefit from the information derived from the model. The paper includes conclusions along with a discussion of future work.

2 Current Approaches for Measuring System Maintainability

Multiple models exist to determine the costs associated with developing systems utilizing COTS components in combination with custom development. The BASIS technique, the COCOTS Model, the COTS Lifespan Model (COTS-LIMO), and the Maintenance Delta are four examples. Three of these models primarily focus on the front-end of the development lifecycle or are focused on cost measurement alone.

This section provides background on the definition of a COTS product. The development phases of a COTS-based software system are defined in order to indicate the applicable program lifecycle phase where each of the available models is used. Following these level-setting sections, current modeling processes are

explored to understand the state-of-the-art in estimating effort associated with developing, deploying and maintaining CBS.

2.1 Commercial Off-the-Shelf Components

COTS, as defined in the Constructive COTS (COCOTS) model, is a component that has the following attributes [6].

- The component is sold, leased, or licensed for a fee that includes fixes for defects.
- The source code for the component is unavailable to the end user.
- The component evolves over time through periodic releases of the product (upgrades) containing fixes and new or enhanced functionality, and
- Any given version of a COTS component will eventually reach obsolescence after which it will no longer be supported by the vendor.

COTS-intensive systems depend on many different COTS applications to provide the business functionality of the overall system. No single COTS product fulfills a significant amount of the requirements [6]. Instead, many products are integrated together to fulfill the business requirements of the system. The CBS design approach relies on glue code or custom code to ensure that COTS products communicate with each other utilizing their documented interfaces. CBS typically deliver functionality more reliably and quickly than custom software development approaches [9,10,11].

In CBS, the developer relinquishes control of the release cycle of the COTS product [12]. A third party controls when bug fixes and new functionality are introduced into the product. The third party's timelines are rarely coordinated with the users of the COTS product. So, users either accept the functionality as is or find other ways to fulfill the business logic that are not provided with the COTS products that are part of the system. Usually this entails creating custom code to provide missing functionality or to work around a bug in the COTS product. Sometimes, to work around a bug in a COTS product, it is necessary to change the way the system is used until a bug fix is released by the vendor. Similarly, a new version of a COTS product may change the functionality of an interface while introducing a fix to a bug (related to the interface or not). The developer may need the bug fix, but the change in an interface may cause rework in the custom code around the COTS product. Since the CBS developer does not control what is in the release from the COTS vendor and what is not, the COTS vendor has the potential to cause rework simply by the way a new release is bundled.

For CBS, the developer can no longer fully coordinate every aspect of the system delivery [13,14] and may have to deal with dependencies that complicate the system deployment without adding value to the system's functionality. These problems are amplified when products from multiple COTS vendors are deployed in the system and incompatibilities between the architectural requirements of the vendors cause conflicts in the deployment of the system [15]. For example, if one product requires Java version 1.5 and another requires Java 1.6 to operate correctly then the overall system must have two different versions of Java installed for both of these products to function correctly. Having two versions of a product in the same system results in complexity around configuration management, deployment and integration of the entire CBS.

In today's large government system architectures, it is typical to have more than 50 COTS products [16,6] and not unusual to have more than 100 COTS products in the final CBS architecture. The architectural dependencies between these products, that is, dependencies on specific product versions, libraries or common variables, complicate the initial deployment, the integration and the on-going maintenance of the system once the system transitions into operations. Often, because of the COTS delivery cycle, the use of COTS products increases the rate of change in a system versus a custom-developed solution [17]. COTS products also increase the risk of change in a production system and contribute to complexity in replacement or upgrade attempts with a resulting failure percentage as high as 70% [18]. The complications associated with the rate of change increase the costs associated with maintaining the CBS, have the potential to strain

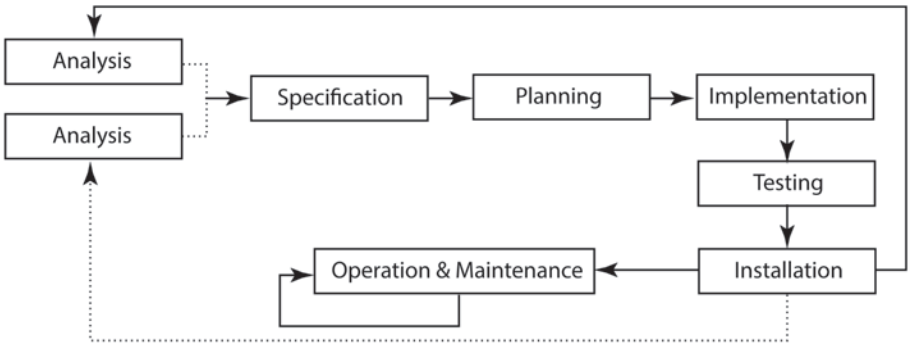

Figure 1: CBS development cycle process (modified from [1]).

the schedule and costs associated with integrating and maintaining the CBS, and have the potential to cause a complete project failure if not accurately estimated and understood.

The problem of maintaining the COTS deployment is not one that is relegated to the O&M phase of the system lifecycle. A typical COTS vendor releases a new version of its product every eight to nine months, and only supports the last three releases of the product [19]. This means that a COTS product may only be supported for 27 months once it is configured into the system unless steps are taken to install an updated version of the product. Changes associated with the product update cycle are in addition to changes introduced by vendor's patch releases that fix bugs. Patch releases only increase the number of changes in the configuration. Today's large system development cycles are often multiple years long with many large systems taking longer than five years to develop [6]. The comparatively brief update cycle of individual COTS products suggests that many products will require updates before the system leaves the development phase of the system lifecycle. And, more importantly, because of this rapid update cycle, a plan to upgrade COTS components in the architecture must be part of a sustainable O&M plan. The vendor upgrade cycle and the need to keep the delivered system in a supported configuration with each individual vendor makes it necessary to consider and understand the architectural interactions of COTS products in the earliest phases of the system design to ensure that the system is maintainable for its planned lifecycle. Additionally, interoperability issues have the potential to paralyze the development of the project if they are not understood and incorporated into the applications requirements [20], so it is helpful to understand these complexities introduced by COTS dependencies early in the project and track them through the project's entire lifecycle.

2.2 Software Development Lifecycle

Several of the existing maintainability metrics for CBS are based on complex measurements of COTS interfaces. Others are based on subjective analysis of the COTS product and how easy it is to use in a development environment. These measures are appropriate for the front-end of the product development lifecycle where software developers are designing and interacting with the products. However, because the maintenance phase of a project is significantly more expensive than the development phase [21], measurements are needed that account for maintainability in the planning, installation, and operations and maintenance phases. Figure 1 highlights the areas of a typical development cycle where maintainability efforts have the most impact on effort for CBS.

2.3 Measuring the Cost of COTS System Developments

2.3.1 The BASIS Technique [22]

The Base Application Software Integration System (BASIS) technique is an integration approach that assists in selecting COTS products and defining the order in which COTS applications should be incorporated into

the delivered system [22]. The technique is used in the first stage of project development - the planning stage - to determine which COTS products introduce the most risk into the system design. Once these products are identified, they are integrated into the system starting with the most complex and progressing to the least complex. An iterative integration process is proposed in the technique implying a spiral development process. The BASIS technique is a small part of the Phase-Integrated COTS (PIC) approach for the entire software development lifecycle. The model considers COTS a portion of the custom software development cycle.

The BASIS technique is used at the start of the design process and is used to select the best COTS candidates to fulfill the requirements of a project. There are three basic evaluation steps to the BASIS approach [22] – (1) how each COTS product fulfills system requirements, (2) the provider's viability and (3) the complexity of the product's external interfaces for integrating with the custom code. The BASIS technique requires extensive knowledge of the COTS products being evaluated. While there is an effort in the technique to account for the order in which COTS products should be integrated into a system, the focus of effort is to minimize the development risk of the project. Specifically, the BASIS technique minimizes the risk associated with developing and integrating custom code and estimating the effort associated with custom code development instead of the effort to maintain the overall COTS installation for the CBS.

The BASIS technique differs from the in-degree mean method proposed herein in that the in-degree mean requires less intimate knowledge of the interfaces associated with each COTS product. Further, the in-degree mean enables the architect to consider the maintenance phase of the program during the design process instead of simply choosing products that are favorable in the development phase. Instead of requiring knowledge of each external interface, the in-degree mean method requires only the installation requirements and the architectural requirements.

2.3.2 COCOTS Cost Model [3]

The Constructive COTS Integration cost model (COCOTS) is a cost estimating tool that recognizes that COTS integration has become a significant portion of modern computing systems. The model attempts to account for the integration of COTS components in the cost estimation of the system design.

COCOTS accounts for four initial integration costs associated with the effort to perform (1) assessment - candidate COTS component assessment, (2) tailoring – work required to configure and integrate the COTS component into the system under development, (3) glue code - the development and testing of any custom integration code needed to plug a COTS component into a larger system, and (4) volatility – the increased system level programming and testing due to volatility in incorporating COTS components. The model accounts for testing in each of the phases. By design, it is focused on the front-end of the development process. The COCOTS model, and its extensions, is used as a software-costing tool; therefore, it is primarily used in the system design and development phases of a program. However, the COCOTS model is being expanded to include maintenance costs in the future [9].

In the assessment process, the COCOTS model defines rating criteria for determining the integration complexity of each COTS product. The rating is subjective and gives a point rating for the complexity associated with a product in each of the following areas: parameter specification, script writing, I/O Report Layout, GUI screen specification, Security / Access protocol initialization and setup, and availability of COTS tailoring tools. The complexity scores for each of the COTS products in a system are combined to determine an overall complexity rating for a project. The model requires an engineer to be familiar with the development interfaces of the COTS product. The knowledge associated with the development interfaces is often gained through years of experience with the product.

The COCOTS model is different from the in-degree mean process because it is estimating the effort associated with development instead of the costs associated with the deployment, integration and maintenance of COTS products. The complexity ratings and integration approaches associated with the in-degree mean assist the costing of the integration effort in a similar way as the COCOTS model assists the development team in determining costs and schedules. The two models are focused on different ends of the program lifecycle as shown in Figure 1. As with the BASIS technique, the COCOTS model requires knowledge

of the development interfaces of each COTS product whereas the in-degree mean process only requires information from the COTS installation manuals and the system CONOPS.

2.3.3 COTS-LIMO [1]

There is a widespread belief in the COTS integration community that the number of COTS components in a CBS has a strong impact on the maintainability of a system. Each COTS product in the system requires unique knowledge to support the integration and maintenance effort. Additional costs are associated with tracking COTS product upgrade roadmaps and licensing costs as well as the support effort required to interface with the vendor to report bugs as they are identified in the CBS. And, the independent release schedule of each COTS product in the system creates a maintenance tail to simply track the compatibility between COTS product versions.

Because of the additional costs associated with CBS, the COTS Lifespan Model (COTS-LIMO) model attempts to identify a break-even point where maintenance costs increase disproportionately to the number of COTS products in a system. According to the model, the break-even point exists regardless of the efficiencies gained. This point is called the maintenance equilibrium. The COTS-LIMO model presumes that it is possible to determine the number of components that exceed this maintenance equilibrium. The COTS-LIMO model acknowledges that the costs associated with maintaining a single COTS component decrease over time as the maintenance staff becomes more familiar with the product. But, even with these efficiencies, the complexity of maintaining multiple COTS products in a single system reaches a point where the costs associated with tracking and maintaining the COTS components exceed those efficiencies gained [5,23] by using COTS products instead of custom-developed components.

The COTS-LIMO is intuitive. As more products are added to a CBS, the maintainer of the system reasonably expects that effort to maintain the system will increase. However, the COTS-LIMO model has one significant drawback; namely, that no process or method currently exists to determine the appropriate number of COTS components that exceeds the maintenance equilibrium for a given CBS design. The model falls short in practical application because it is not possible to determine when there are too many COTS products in a CBS. So, it is not possible to make architectural decisions about whether to fulfill a system requirement with a COTS product or custom development.

2.3.4 Maintenance Delta [24]

The theory of the Maintenance Delta asserts that the Power-Law distribution is a standard for the number of architectural interactions in a system and, therefore, represents an ideal on which the maintainability of real-world systems can be measured.

In network theory, scale-free networks hold a unique position in that they have been found to describe many large networks. This holds true for both naturally occurring networks (cellular metabolism) and for man-made networks (learning networks, the Internet, etc.) [25,26]. The scale-free network is a network where the degree distribution of the nodes follows the power-law distribution [27] – a probability distribution, pk, of degree k with the form

$$p_k = Ck^{-\alpha} \tag{1}$$

for integer values of k where $k_{min} > 0$, where C is a normalization constant, and α is a constant parameter of the distribution known as the exponent or scaling parameter. α typically lies in the range $2 < \alpha < 3$ [28]. The number of nodes in each dependency group differs with the size of the network; therefore, the scale-free model is intended to accommodate networks of various sizes.

The maintenance delta is the numerical difference between the distribution of a CBS and a power-law distribution of similar size. A CBS with a distribution that is below, or less than the power law distribution, would have a negative maintenance delta indicating that the level of effort to maintain the system is less than normal. A CBS with a distribution that lies above, or has more interactions than a similarly sized

scale-free network, has a positive maintenance delta and, thus, requires more effort than normal to maintain the COTS installation.

The maintenance delta is intended to be a comparative measure of the maintainability of two systems. By deriving the maintenance deltas of the two competing architectures and comparing the results, it is possible to make design decisions based on the maintainability of the overall solution. The maintenance delta and the in-degree mean process both explore maintainability for the O&M phase of the CBS development lifecycle.

3 Maintainability and Complexity of CBS

Like software maintainability, COTS maintainability is based on complexity; therefore, maintainability should be accounted for by the interactions between the components. A process for measuring the maintainability that combines the unique qualities of a COTS-based system along with the interactions between those products is needed for CBS to appropriately measure their maintainability. The concept of maintainability is focused on the operations and maintenance phase of the program and focused on the maintainers of the system, not the consumer of the system's functionality. The focus on the O&M phase is distinctly different than software models where most effort is focused on the development phase of the program (see Figure 1). Similarly, other software models focus on the effort for software engineers to maintain the custom code, including glue ware, of the system instead of the effort required to maintain the components for which there is little visibility into the internal structures of the software and little influence on the delivery roadmap [29,16]. For these reasons, a CBS maintainability measure should focus on the interactions between the components and the effort associated with changing the components and the effort required by maintenance engineers to upgrade and maintain these components.

The in-degree mean model presented herein complements the existing models by filling a gap in the available models. By covering the aspects of system deployment, the integration effort associated with that deployment, and the maintenance effort in the Operational and Maintenance (O&M) phases, the model adds a more detailed investigation of an area of the CBS lifecycle that has been overlooked in other models.

3.1 Process of System Maintainability

The maintainability of CBS is defined by how much effort is expended in the various phases of the program; however, as the O&M phase is typically the longest in the program lifecycle, the effort associated with this phase has the highest impact on the maintainability of the system. Maintainable systems minimize the effort required to achieve five maintenance activities associated with COTS-based systems [14]

1. Product reconfiguration
2. Testing and debugging
3. System monitoring
4. Enhancing user-level functionality
5. Configuration management

Table 1 shows these five maintenance activities along with a description of common general activities that comprise each category. Product reconfiguration is the replacement of given product with an upgrade to the same product or replacement with another product that performs a capable functionality. The other attributes of the systems that require effort – testing and debugging; system monitoring; enhancing user level functionality; and configuration management - are somewhat self-explanatory. Other factors influence the cost of maintaining COTS-intensive systems [6]. However, each of the influencing factors can be categorized into one of the five identified maintenance activities.

Extensive research has been done on complexity and on measuring complexity of software development [30,31,32,33,34]. Much of this research, particularly around component based software development, is

Table 1. Maintenance activities that drive significant effort in CBS.

Maintenance Activity	Description	Implication of COTS usage
Product Reconfiguration	Updating COTS products with new versions Replacing products with competitors offerings Adding / deleting products as requirements evolve	Update schedule determined by COTS vendors Updates uncoordinated between vendors Features, bug-fixes and adaptations driven by market force of COTS products rather than system requirements
Testing and Debugging	Identifying causes of failure Running tests Monitoring system performance and resource utilization Logging system behavior and activity	COTS products are black-box: no access to source code Documentation often incorrect or incomplete Done in collaboration with COTS support organizations COTS suppliers do not accept responsibility for problems without proof (and maybe not even then)
System Monitoring	Logging system behavior Analyzing logs for failure, performance problems, etc.	Lack of visibility into behavior of individual COTS products
Enhancing User-Level Functionality	Modifying functionality as requirements evolve	No / limited access to source code of systems Dependent on tailoring facilities provided with COTS product Glue code and wrappers are major means of tailoring
Configuration Management	Tracking available versions of COTS products Tracking change history of products Recording set of compatibilities and incompatibilities between sets of products Tracking current configuration of products at each deployed site Tracking change history of products of each deployed site Managing license and service agreements for each product	Versioning controlled by the COTS distributor Licensing and support agreements must be managed Management of COTS configurations Compatible versions of COTS products must be determined

pertinent to this work. With component-based software development, the interaction between components becomes the focus for creating maintainable solutions instead of the internal workings of each individual component [35]. However, this present study aims to be complimentary to these models for custom software development and cover an area of maintenance that is not typically included – that associated with maintaining the COTS products.

3.2 Complexity

3.2.1 Identifying Dependencies

In a large system, the individual products combine to deliver the system's defined functionality; however, to deliver the functionality, the components must be integrated into a single system. Each COTS product requires additional components and characteristics to function as desired in the system. These requirements are defined as COTS dependency attributes [36] and include such items as the need for a specific version of another COTS product or the definition of an environment variable in the system where the product is installed. Behavioral dependencies - the interaction between the two components that may only be certified for specific versions of the two products [14] - are included in COTS dependency attributes.

Architectural mismatch is another similar area of study that has received much attention in the literature [37]. As with the COTS integration effort [38,12,15], the architectural mismatch research has focused on the front-end development of programs. This research has generally focused on the interfaces between

COTS products, the interfaces that are used by software developers to interact with the COTS products, and the software architecture in which the components are functioning. A fourth area, assumptions about the construction process, is closely linked to dependency attributes in that it recognizes that software components (including COTS components) have underlying assumptions about the order in which the system is built or about previously existing capabilities that must be in place for the components to function correctly. However, as the proliferation of COTS products increases in computing architectures, the architectural mismatches caused by requirements from disparate COTS products is an area that has not been covered by previous research.

Architectural dependencies and dependency attributes are traditionally acknowledged as creating interactions in a CBS. But, architects also create interactions in the CBS by choosing which products fulfill specific requirements and by defining the system's concept of operations (CONOPS). These two types of interactions contribute to complexity in testing and debugging the system by creating additional interactions within the system [39]. For instance, if a single product is chosen to fulfill five different functional requirements in the system, changing that product requires more functional testing than if the product only fulfilled a single functional requirement.

The concept of operations creates interactions by introducing interactions between components that are not necessarily intended by the vendor [24]. One COTS product may not require another to install and operate, but the two may be required to interact and provide the required functionality of the system. This interaction is not covered by the architectural or behavioral dependencies and is, therefore, missing from the current models that are in use. Combined, architectural dependencies, COTS dependency attributes, architectural mismatch, and the system's concept of operations create a more complete view of the architectural interactions in the CBS.

Architectural interactions themselves create complexity in the CBS that must be managed. And, because each COTS product vendor follows its own independent development and upgrade schedule, the innate complexity in a CBS is worsened by the need to upgrade individual components on their own individual time scales to remain in a supported configuration. The time-dependent combinatorial complexity created by multiple independent upgrade paths must be eliminated from the system in order for the maintenance lifecycle of the system to be successful and avoid degenerating into a chaotic state [40]. This emergent behavior is one of the aspects of a complex system [8,40,41] and must be anticipated and understood in the design phase of the CBS in order to achieve the appropriate maintenance equilibrium of the system in the deployment and maintenance phases. Without appropriate sustainment, the system has the potential to devolve into disorder [42]

3.3 Complexity

3.3.1 Identifying Dependencies

In a large system, the individual products combine to deliver the system's defined functionality; however, to deliver the functionality, the components must be integrated into a single system. Each COTS product requires additional components and characteristics to function as desired in the system. These requirements are defined as COTS dependency attributes [36] and include such items as the need for a specific version of another COTS product or the definition of an environment variable in the system where the product is installed. Behavioral dependencies - the interaction between the two components that may only be certified for specific versions of the two products [14] - are included in COTS dependency attributes.

Architectural mismatch is another similar area of study that has received much attention in the literature [37]. As with the COTS integration effort [38,12,15], the architectural mismatch research has focused on the front-end development of programs. This research has generally focused on the interfaces between COTS products, the interfaces that are used by software developers to interact with the COTS products, and the software architecture in which the components are functioning. A fourth area, assumptions about the construction process, is closely linked to dependency attributes in that it recognizes that software components (including COTS components) have underlying assumptions about the order in which the system is built or about previously existing capabilities that must be in place for the components to

function correctly. However, as the proliferation of COTS products increases in computing architectures, the architectural mismatches caused by requirements from disparate COTS products is an area that has not been covered by previous research.

Architectural dependencies and dependency attributes are traditionally acknowledged as creating interactions in a CBS. But, architects also create interactions in the CBS by choosing which products fulfill specific requirements and by defining the system's concept of operations (CONOPS). These two types of interactions contribute to complexity in testing and debugging the system by creating additional interactions within the system [39]. For instance, if a single product is chosen to fulfill five different functional requirements in the system, changing that product requires more functional testing than if the product only fulfilled a single functional requirement.

The concept of operations creates interactions by introducing interactions between components that are not necessarily intended by the vendor [24]. One COTS product may not require another to install and operate, but the two may be required to interact and provide the required functionality of the system. This interaction is not covered by the architectural or behavioral dependencies and is, therefore, missing from the current models that are in use. Combined, architectural dependencies, COTS dependency attributes, architectural mismatch, and the system's concept of operations create a more complete view of the architectural interactions in the CBS.

Architectural interactions themselves create complexity in the CBS that must be managed. And, because each COTS product vendor follows its own independent development and upgrade schedule, the innate complexity in a CBS is worsened by the need to upgrade individual components on their own individual time scales to remain in a supported configuration. The time-dependent combinatorial complexity created by multiple independent upgrade paths must be eliminated from the system in order for the maintenance lifecycle of the system to be successful and avoid degenerating into a chaotic state [40]. This emergent behavior is one of the aspects of a complex system [8,40,41] and must be anticipated and understood in the design phase of the CBS in order to achieve the appropriate maintenance equilibrium of the system in the deployment and maintenance phases. Without appropriate sustainment, the system has the potential to devolve into disorder [42].

3.3.2 Mapping Dependencies

The Design Structure Matrix (DSM) has been in use for decades to show dependencies between tasks in design and manufacturing of large engineering systems [43]. The DSM has proven to be a valuable tool in understanding and managing complexity in sophisticated design projects in the automotive and other industries. Recently, the DSM has been used to create the foundation of a network to study the task interactions in a product development process [44] and modularity in software development [45]. Based on this work, it is straightforward to leverage the DSM tool to map the architectural interactions and assist in managing the complexity associated with maintaining a CBS.

Figure 2 shows an example DSM. The numbers along the top and side of the DSM represent COTS products in the system – a total of 13 in this case. Each product is listed in the same order on both sides of the DSM resulting a square matrix. A mark in the square where two products intersect indicates a dependency that the COTS product in the row has on the COTS product designated in the column. For example, in Figure 2 product 1 is dependent on itself and product 12. Similarly, product 3 is dependent on products 1, 4, 5, 6, 7, 8, 9, 10 and itself. Product 2 is unusual in this DSM because the only dependency it has is on itself. This relationship is directed. It may or may not be true that the product in the column has a dependency on the product in the row. For this reason, the DSM is not symmetrical once it is fully populated. Each product has a dependence on itself; therefore, the diagonal is fully populated in the DSM. Looking at Figure 2, the row for COTS product [1] shows the dependency that the product has on itself in column [1] and another dependency on the COTS product in column [12].

In CBS, DSMs assist in scheduling decisions and identify architectural dependencies. They also provide a visual representation of the system. It is sometimes convenient to transform the DSM into a network or graph where the system can be analyzed using graph theory [45]. When the DSM is converted to a

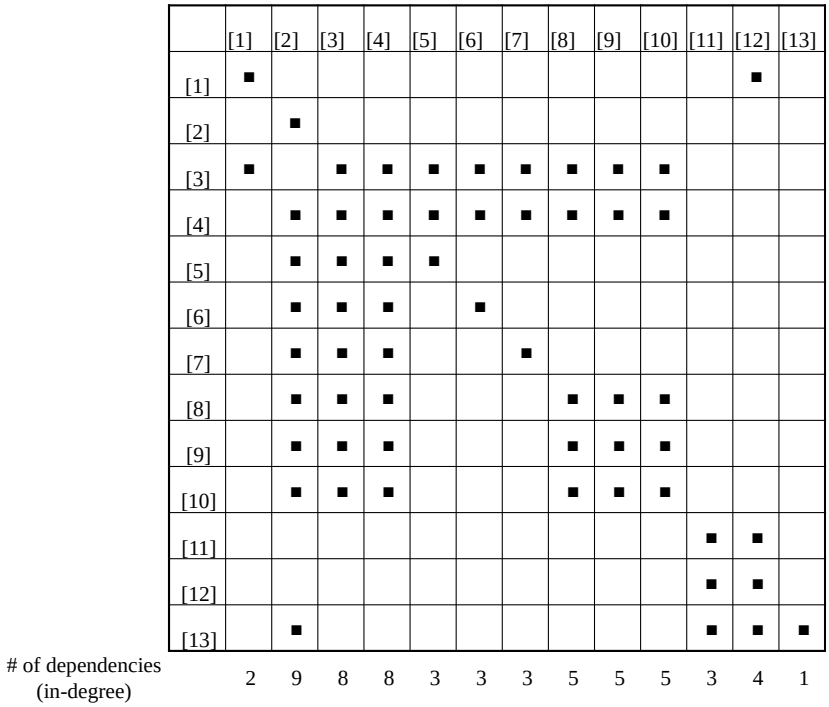

Figure 2: DSM showing interactions between COTS products and the in-degree for each product.

network, each connection other than the link the COTS product has to itself is a link to another node in the network.

Graph theory is used frequently in computer science and system design to represent software and architectural designs, computer networks, and even the Internet [46,47]. Graph theory is a useful mechanism to study networks as it offers a common language to label and represent the network as well as mathematical notions and operations with which network properties can be quantified and measured. The graph, or network, consists of nodes and connections between the nodes. Degree, degree sequence, and degree distribution are three of the common graph theory attributes that give information about the network [48]. Because directed graphs give information about the relationships between two nodes that is not available in undirected graphs, directed graphs present a more accurate representation of the structure of the network [49,50]. In a directed graph, *in-degree* refers to the links coming into a node, and *out-degree* refers to the number of links coming out of a node. In the DSM for CBS, the in-degree refers to the dependencies that other COTS components have on a node. Out-degree refers to the dependencies that a node has on other COTS components.

3.3.3 Measuring Complexity and Maintenance Effort

Looking at Figure 2 again, the in-degree for each COTS product in the system is obtained by simply summing the number of entries in each of column where each column represents a single COTS product. The in-degree is the number of products that depend on the COTS product identified by the column in the DSM. Therefore, some products have an in-degree of 1 because no other products depend on that product to provide functionality in the system. Conversely, each product depends on an operating system, so the operating system column has multiple in-degree dependencies. Figure 2 identifies the in-degree for each COTS product, computed and shown at the bottom of each column.

The arithmetic mean of the in-degree of the nodes, or average in-degree, is a characteristic of the network. The average, shown in (2), defines the *in-degree mean*. The mean (average) in-degree, c, in a directed graph is

$$c = \frac{1}{n} \sum_{i=1}^{n} k_i \tag{2}$$

where the in-degree of node i is denoted by k_i and n is the number of nodes in the network [51]. The arithmetic average is straightforward to calculate and it incorporates the contribution from every node in the network; however, strong outliers may heavily influence the average [52]. Components with many connections require significant effort to test and maintain compatibility with all of the other products with which they interact. Also, in dynamic networks where nodes are continuously being added (e.g., the Internet), the average degree of the network appears to be increasing [53]. In the case of CBS, the architecture of the system is static, so the average degree is constant for each network under evaluation.

4 Measuring Complexity and Maintainability

The in-degree mean model is proposed as a predictive measure of the effort required to maintain a CBS through the O&M phase of the CBS development cycle. To empirically assess the model, data for CBS were gathered from multiple sources [54].

4.1 Sample Population

The population of systems available for assessment with the model included CBS from one large company (> 10,000 employees), two medium sized companies (> 100 employees and < 10,000 employees) and one small company (< 100 employees). Projects selected for this study were not required to satisfy criteria other than their sponsor's willingness to participate. Therefore, nothing was known up-front about the number of COTS products in each system or the effort required to maintain each system. Using multiple data sources strengthens the study by eliminating engineering design bias introduced by standardization in an organization [55]. A request for participation was sent to program managers and leaders from each of these organizations. During introductory conversations, each organization provided an approximate number of systems on which they would provide information. 53 responses were expected; however, only 17 responses were received.

While 17 represents a significant reduction from the expected 53 systems, each organization experienced unique challenges in providing the details of the systems they had architected and delivered. Some sources were concerned about program security and releasing company-sensitive information and were unable to provide the requested information. Others suffered from staff shortages and were not able to provide the information because of timeline constraints from other deadlines. One company outsourced the entire Information Technology (IT) department and did not retain the expertise required to provide the information about their systems.

From the 17 responses, three were eliminated during the evaluation period because the systems were cancelled before entering the O&M phase of the program or because the supporting personnel were not available to participate in the questionnaire. One system was eliminated because two different contractors developed and maintained the system. This system is different because in all the other systems in the response set, the same contractor developed and maintained the CBS. Instead of introducing another independent variable associated with the separation of the development and maintenance contracts, the system was excluded. An item was added to the future work (Section 7.4) relative to adapting the model to address cases where the developer and maintainer of the CBS are different.

After these exclusions, the model was tested with a sample size of 13 systems. All of the evaluated systems except one were in the operational stage of the lifecycle. The one remaining system was being decommissioned; therefore, it had already been through its operational lifecycle. The final sample set

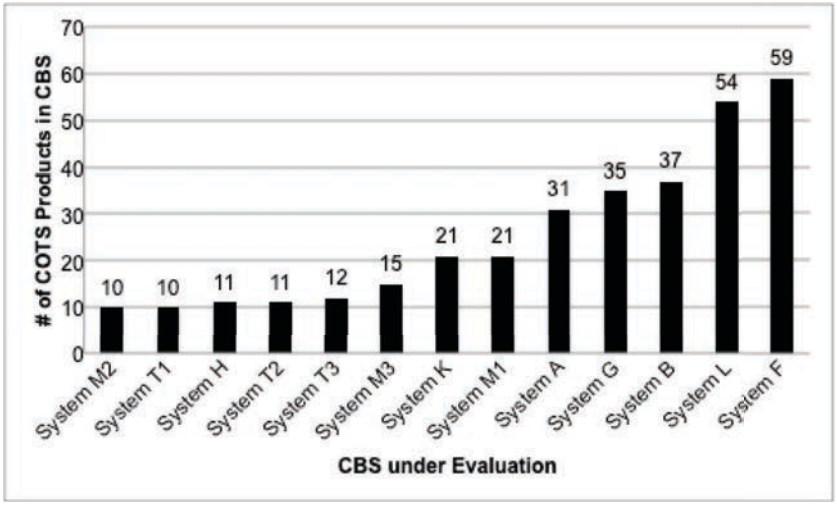

Figure 3: Number of COTS products in each system in the sample set.

ranged from ten to 59 COTS products (see Figure 3) representing systems from small to large numbers of COTS products [6].

The sample size is larger or similarly sized to other published studies related to CBS maintainability. For instance, in IEEE, only four published empirical studies were identified, these having two [56], five [16], eight [48] and thirteen [13] systems surveyed.

The following sections explain how the dependencies for the 13 systems in this study were derived (a measure of complexity) and how effort to maintain these systems was evaluated (a measure of maintainability).

4.2 Identify Complexity in CBS

To establish architectural interactions in the subject CBS, a request was sent to system architects and program engineers asking for a list of COTS components in the system for which they were responsible. A follow-up interview to understand the CONOPS associated with the system and to identify any additional dependencies was also requested as part of the initial contact with the system subject matter expert (SME). This part of the process was challenging for some of the programs because of concerns with revealing sensitive program information or because knowledgeable personnel were no longer available. This suggests that for operational programs, the information required to create the in-degree mean model may be difficult to obtain. However, for systems under development, knowledgeable personnel should be readily available for determining the information required for the model.

For most of the projects, the follow-on interview lasted less than one hour. Architectural data flows were helpful in ensuring that all dependencies were identified and to refresh the program SME on project functionality that was used less frequently.

4.2.1 Dependencies in Existing Documentation

After receiving the COTS list from the program representative, COTS installation dependencies were identified by reading the vendor documentation for each of the products in the CBS. Many installation and configuration dependencies are recognized in literature [57,14] and are straightforward to define. With this process, the mapping of installation and configuration dependencies relied on the accuracy of vendor documentation to ensure that it was complete and up-to-date.

The availability and quality of vendor documentation varied significantly as there is no standardization in

Installation Requirements

The following table outlines installation requirements for Hummingbird connectivity products:

Product	Operating System	Disk Space	Other Requirements
Exceed	Windows 95 Windows 98 Windows Me Windows NT (service pack 4 or later) Windows 2000	78MB	Winsock compliant TCP/IP
Exceed XDK		219 MB	Winsock compliant TCP/IP Microsoft Visual C/C++ (MSVC) 4.2 or later, for X Client development
Exceed 3D		6 MB	Exceed – to Open GLX Exceed XDK – to display Open GLX Microsoft Visual C/C++ (MSVC) 4.2 or later, for X client development
Exceed PowerSuite		96	Winsock compliant TCP/IP An assigned IP address and the ability to communicate with other computers on the network (Windows NT) A HOSTS file if a domain server is not available

Third Party Software

Certain third party software must be installed to run some Hummingbird products. The Sun Java Runtime Environment (JRE) is required to run Java programs.

For example, you must install JRE before installing the following product sub-features:

- Systems Administration—Jconfig (client)
- Systems Administration—Jconfig Daemon
- Exceed (Tools)—Xdis

Hummingbird Master Setup lets you install third party add-on(s) such as:

- Adobe Acrobat Reader
- Sun Java 2 Runtime Environment
- Microsoft SNA Server (for Windows NT/2000)
- IntranetWare for SAA Client

Figure 4: Example installation prerequisites for Hummingbird Exceed (COTS product) [47].

installation guides for COTS software [58]. When COTS vendors document their installation dependencies, this takes many forms. The installation guide is the most common place where vendors document installation prerequisites. Figure 4 shows how one COTS product, Hummingbird's Exceed – a COTS application that lets the user access Linux or UNIX applications from a Windows-based workstation – documents its dependencies on the operating system and on the Java Runtime Environment.

Hummingbird's Exceed notes dependencies in at least two ways. First, depending on the product used in the system (four different products are covered in the embedded table), the operating system is noted in the second column of table and additional requirements are noted in the fourth column. Because multiple operating systems are supported, it is necessary to know the baseline operating systems, including versions, used in the system under evaluation. Once the operating system is known, a dependency is noted in the

DSM at the intersection of the row for Exceed and the column for the operation system. The same process is used to document the dependency on TCP/IP and Microsoft C/C++ (from the fourth column in the embedded table in Figure 4) in the DSM. Additional third party dependencies are noted depending on how the product is to be used in the system.

The "Third Party Software" portion of Figure 4 indicates another dependency on the Java Runtime Environment (JRE). This dependency is written out instead of included in the embedded table; however, the dependency must still be captured in the DSM. It is important to recognize that vendors document installation dependencies in multiple ways even within a single vendor's documentation.

For the CBS in the dataset, determining the dependencies took about one hour per product. The vendor documentation that lists the installation dependencies must be located. The Internet is a valuable resource for these documents as most vendors make their documents publically available. Once the appropriate manual is located, the installation dependencies are relatively easy to find in the manual. They are generally called out in the Table of Contents or found with keyword searches. Most vendors make installation dependencies clear even when the product installation is dependent upon another company's product. For instance, many products are dependent on a database for storing information. In these cases, the vendor is clear on which vendor's databases are supported along with the required versions of those databases.

Many products are commonly used across projects. For instance, Internet Explorer and the Oracle database are typical components in many systems. The commonality of these components accelerated some of the investigation of dependencies but only when the same versions of the components were used between systems. When different versions were used, it was not possible to leverage previous work because of the potential mismatch of the versions of dependencies.

4.2.2 Dependencies from the CONOPS

In addition to dependencies introduced by the COTS products themselves, CBS also include dependencies derived from the system CONOPS. Combined, these are called architectural interactions. The dependencies from system CONOPS were derived during an interview with the system architect or other SME who was knowledgeable of the overall design principles and concepts of the system. During the interview, the architect identified dependencies that were introduced by the system data flows or system design. Such dependencies are sometimes related to the way application workflow states are preserved or to concepts related to application fault tolerance and failure recovery. Similarly, in many systems, the database is used to preserve processing state information for web applications. It is possible that no vendor's installation guide recognizes this dependency; however, for the system to function as the architect has designed, the dependency between the products must be recognized and maintained throughout the lifecycle of the system. In the DSM, these dependencies are noted with a mark on the web application row in the column for the database. Dependencies derived from the CONOPS are unique to the design of the CBS; therefore, they can only be determined by the architect and will not be documented in the vendor requirements for each individual product.

The process of documenting dependencies between COTS products (whether installation or CONOPS) continued until all of the known dependencies were identified with a mark in the DSM. The diagonal of DSM is always marked as the dependency that each product has on itself. Once complete, the DSM documents all of the architectural interactions of the CBS and becomes the basis of the model for determining the maintainability of the CBS.

Because there is some ambiguity in the dependencies that are known at the beginning of a project, the DSM is a living model of the system. As new dependencies are introduced or knowledge of the existing dependencies matures, the DSM is updated to increase the fidelity of the model.

4.3 Real-World Assessment of Effort – the Survey Questionnaire

In order to validate the accuracy of the in-degree model of maintainability is it necessary to compare the measure against the effort required to maintain actual deployed systems. Ideally, the effort associated with

integrating and maintaining the COTS products in a system would be captured by some direct means such as charge numbers, billing information, etc.; however, in surveying commercial companies, academia, and government organizations, none of the organizations kept billing metrics at the granularity needed to isolate the maintenance effort associated with only the COTS solution. Therefore, a survey of system architects was determined to be the best means to determine the perceived effort required to maintain each COTS based system. While not as accurate as direct measurement based on billing information, *perceived effort* gives a notional measure of the effort to maintain a CBS.

The construction, execution, and evaluation of the survey questionnaire was accomplished through several steps.

4.3.1 Constructing the Questionnaire

1. Closed ended questions
2. Clear items
3. Only single questions (no double-barreled questions)
4. Only relevant questions
5. No negative questions
6. Non-biased items and terms
7. Short answers (when possible)

All survey items were created to evaluate the perceived effort associated with maintaining the COTS components of a CBS as defined in previous research [14]. As there are five major areas that contribute to the effort – Product Reconfiguration; Testing and Debugging; System Monitoring; Enhancing User-Level Functionality; and Configuration Management - the questionnaire was organized with five major question groups corresponding to each area of effort. Within each major question group, a single survey item addressed each of the individual maintenance activities. Combined, these items comprise a measure of the perceived effort associated with each major category. A sample section of the survey questionnaire is shown in below.

Your role on the project: (e.g., architect, COTS manager, etc.)
Current phase of project: (e.g., development, production, integration, etc)
How long has the project been in this phase? (years or months)

Estimate the effort associated with activity as it relates to the COTS (Commercial Off-the-Shelf Products) in the architecture. If the program or project has not experienced the described type of COTS product change, please mark N/A instead of estimating the effort.

Product Reconfiguration

1. Updating COTS products with new versions. i.e., updating Oracle 10.1 to 11.0, or Internet Explorer from version 7 to 8.
   Effort (from 1-10, or N/A)

2. Replacing a COTS product with a competitor's offering. i.e., replacing DB2 with Oracle.
   Effort (from 1-10, or N/A)

3. Adding or removing COTS products from the architecture as requirements evolve.
   Effort (from 1-10, or N/A)

Testing and Debugging

1. Identifying causes of failure in the system. i.e., isolating the fault to a COTS product and logging the case with the vendor.
   Effort (from 1-10, or N/A)

2. Running tests to validate requirements and verify changes in configurations. i.e., effort associated with testing requirements fulfilled by COTS products.
   Effort (from 1-10, or N/A)

3. Monitoring system performance and resource utilization of COTS products. i.e., validating system performance requirements and resource utilization of COTS products.
   Effort (from 1-10, or N/A)

4. Analyzing logs associated with COTS products to determine system behavior and activity. i.e., utilizing COTS logs to debug or determine system functionality.
   Effort (from 1-10, or N/A)

System Monitoring

1. Logging system behavior. i.e., the effort associated with finding logs associated with COTS products.
   Effort (from 1-10, or N/A)

2. Analyzing logs for failures, performance problems, etc. i.e., the effort associated with determining information about a particular COTS product's failure and performance based on the logging provided by the vendor.
   Effort (from 1-10, or N/A)

Enhancing User-Level Functionality

1. Changing system functionality as requirements evolve utilizing the current COTS in the system. i.e., delivering additional or different system requirements with the COTS products that are already part of the system.
   Effort (from 1-10, or N/A)

Configuration Management

1. Effort associated with tracking the available versions of COTS products. i.e., effort associated with determining when newer versions of a COTS product are released.
   Effort (from 1-10, or N/A)

2. Tracking the change history of COTS products. i.e., tracking the versions placed into the system including reasons driving the change.
   Effort (from 1-10, or N/A)

3. Recording set of compatibilities and incompatibilities between sets of products. i.e., determine compatibility between a new version of a COTS product and the existing versions of the other products in the system. e.g., determining the (in)compabilities of all the COTS products in a system with a new version of the operating system.
   Effort (from 1-10, or N/A)

4. Tracking current configuration of products at each deployed site. (assumes that multiple instantiations of the system exist and that it is possible to have different configurations at each location).
   Effort (from 1-10, or N/A)

5. Tracking change history of products at each deployed site. (as with #4 above, assumes that there are multiple instantiations of the system and that each system can be updated / changed independently).
   Effort (from 1-10, or N/A)

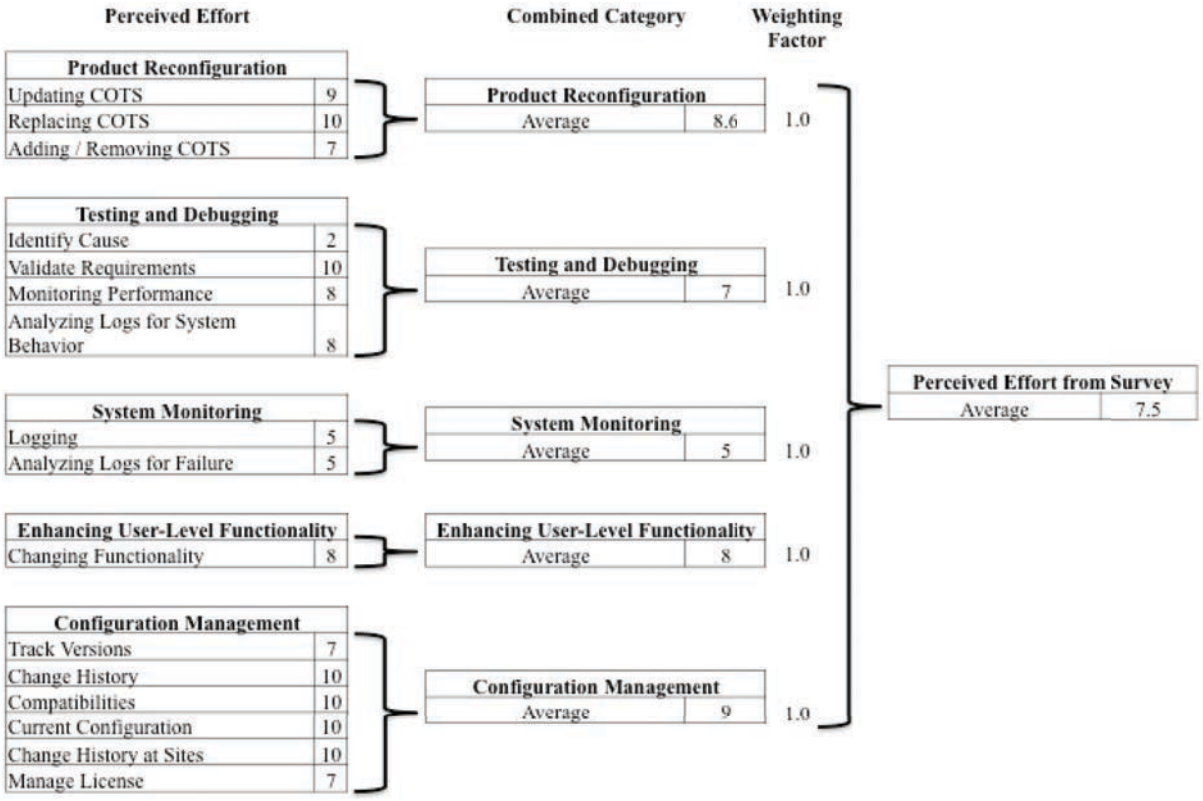

Figure 5: Method for computing the effort for maintaining a CBS from the questionnaire.

6. Managing license and service agreements for each product. i.e., tracking the licensing and service agreements with each of the products in the project configuration. (assumes that at least one COTS product in the configuration requires a license and/or service agreement.)
   Effort - (from 1-10, or N/A)

Additional Notes / Thoughts: (Please indicate the question the note pertains to.)

The survey items were scored using a single numerical rating of the effort in each category (a Likert scale [60,61]). Short answers enabled quicker analysis of the data from the questionnaire and allowed the respondent to easily rate the system while maintaining the integrity of the measure. An ordinal scale (where 1 represented minimal effort and 10 represented maximum effort) was used to compare the perceived effort of maintaining of the evaluated system. Each item allowed the respondent to answer with a "not applicable" (or N/A) response in the event that the system under evaluation did not experience any effort associated with a maintenance activity. The "N/A" response allows each of the items in the questionnaire to be exhaustive – enabling all known answers to the question [62]. The "N/A" response is particularly relevant in systems that only have a single instantiation or where the maintainers of the system have not encountered error conditions where COTS vendor support is required.

The face validity of the survey items was based on previous research identifying all of the areas in a CBS that contribute to the effort in maintaining the system [14] and by having three SMEs analyze the survey instrument [63] to ensure all areas of effort associated with maintaining the COTS components were captured. The questionnaire was initially issued to a small sample of respondents to improve upon its clarity and readability. After two revisions, the third and final version was submitted to all of the

Table 2: Summary of results.

Systems ↓	Perceived Effort (from questionaire)	Normalized Perceived Effort ((Perceived Effort-5)/10)	In-Degree Mean (from DSM)
System A	4.10	-0.09	2.68
System B	7.53	0.25	4.65
System F	5.40	0.04	3.39
System G	2.75	-0.23	2.69
System H	4.05	-0.10	2.45
System K	4.75	-0.03	2.29
System L	5.67	0.07	3.04
System M1	1.67	-0.33	2.76
System M2	1.42	-0.36	2.30
System M3	2.29	-0.27	2.80
System T1	4.75	-0.03	3.30
System T2	4.65	-0.04	2.64
System T3	5.50	0.05	3.17

respondents.

4.3.2 Human Subjects

To help ensure that the respondents had sufficient experience in the field to make an accurate assessment of the effort to maintain the CBS, only senior engineering leaders participated in the survey. This minimized the chance of a respondent having a limited experience on which to evaluate the system under test. Each of the engineers or sustainment personnel completing the questionnaire had over 15 years of experience in the field with five of the 13 having more than 25 years of experience.

4.3.3 Evaluation (scoring) of the Survey

The result from each response in the survey was treated as an ordinal value. While a rating of 7 means more effort was exerted in maintaining the system as compared to a rating of 6, it is not valid to attempt to distinguish gradients between the two scores [64]

4.3.4 Maintainability Index

The overall maintainability index of each system was obtained by averaging the scores from the five maintenance activities known to comprise effort to maintaining CBS (see Figure 5). When computing the maintainability index, all items were weighted equally [7]; therefore, the overall rating of the perceived effort associated with each maintenance activity was obtained by averaging the responses in that activity's area.

The computation of perceived effort associated with system maintenance activities had to accommodate the "not applicable" (N/A) response. The approach to dealing with missing data is not standardized and is unique to every situation [59]. In this study, the maintainability index for the system where one area of effort was not pertinent was computed as if the question did not appear in the survey. Essentially, the number of survey items for the specific maintenance activity was decremented and the item simply was not scored in the average. While this has the side effect of increasing the significance of the effort of those items in the maintenance activity that do contribute to the effort in maintaining the CBS, it is the most straightforward way of accommodating those systems that do not have every aspect of maintenance involved in their sustainment. The option of substituting a nominal value (e.g., 5 since a rating of 5 represents typical effort) for the question with the "N/A" response was considered; however, this artificially changes the effort estimation by adding a contribution for an activity that is not part of the system. Alternately,

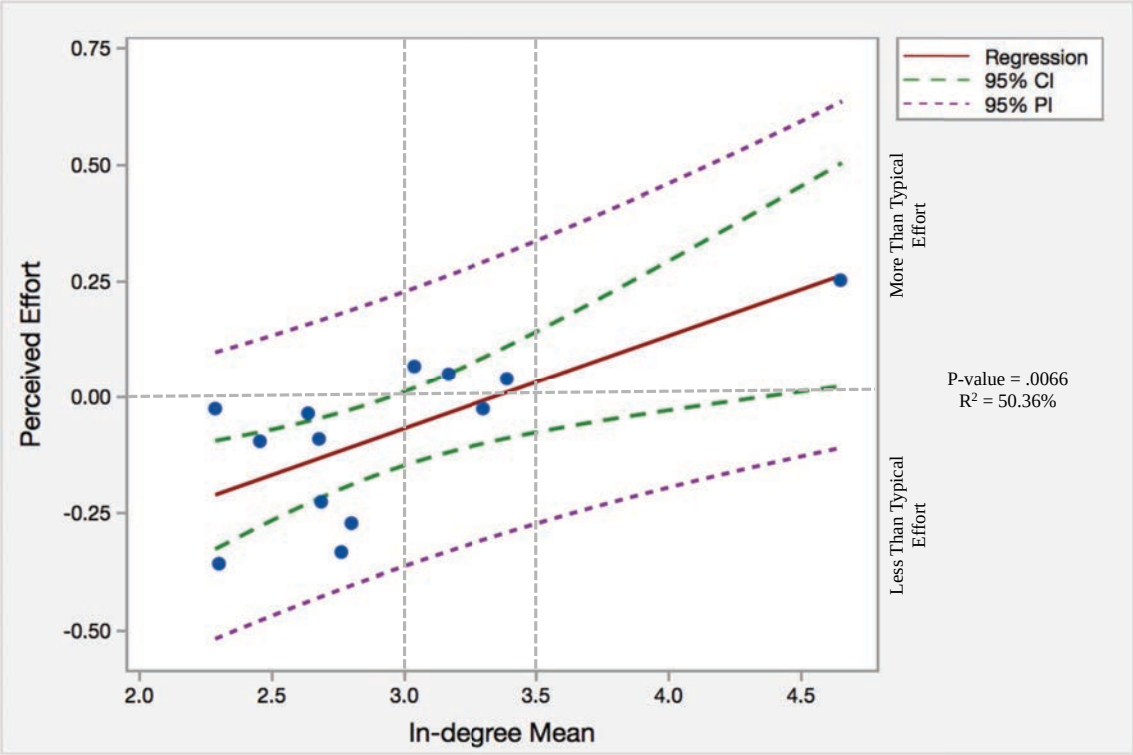

Figure 6: Plot demonstrating a relationship between the perceived effort and the in-degree mean.

dropping the item for all of the systems under evaluation was considered; however, because the sources of effort in maintaining systems is already established in literature and many of the systems under evaluation reported effort associated with every aspect of maintainability, it did not seem reasonable to exclude effort associated with these activities for those systems for which effort was expended.

5 Results

The results from the survey questionnaire (a measure of perceived effort) and the in-degree mean computation (a measure of CBS complexity) are summarized in Table 2. The perceived effort (maintainability index) score has been normalized to transition normal effort to 0 on the scale; therefore, negative effort represents effort that is less than normal to maintain a system, and positive effort indicates a system that requires more effort to maintain than normal.

The model for the in-degree mean asserts that the mean of the in-degree of the nodes in a CBS predicts the effort required to maintain the CBS in the operational phase of the system. To better visualize this relationship, it is helpful to plot the computed in-degree mean from each CBS as it relates to the perceived effort derived from the survey for the same system (Figure 6). The regression model in Figure 6 suggests a statistically-significant relationship exists between the two measures of effort to maintain the CBS. The p-value for the relationship is 0.0066, within the significance level of 0.05, and the coefficient of determination (R2) value is 50.36%.

Not withstanding evidence that a predictive relationship exists, it can be observed that the 95% confidence interval bands become wider as the in-degree mean increases, and the 95% predictive interval bands are quite wide. This bandwidth relates to the size of the dataset, the fact that the dataset contains few systems having high in-degree mean values, and that measurements of perceived effort are noisy. As

has been noted, empirical studies of this type are rare and face many challenges. The promised dataset for this study which began as 53 systems yielded only 17 systems, 13 of which were useable. Even so, the dataset for this study is larger than or similar in size to datasets from other published empirical studies. The same can be said for the number of systems with high in-degree mean values – such systems were not well represented and only one made it to the final dataset. While this particular system does influence the empirical model, the data reveal no evidence to suggest the measured parameters for this one system are anything other than valid.

To further explore the strength of the relationship, Kendall's tau coefficient was calculated to assess the association based on ordinal ranking of the data. Kendall's tau is less sensitive to the magnitude of outlying values and has more robust statistical properties. In this study, Kendall's tau was calculated as 0.374, with a p-value of 0.087 for the null hypothesis test, H0: tau = 0. The p-value is marginally significant, but if we accept this, Kendall's tau suggests a relationship exists between the in-degree mean and perceived maintenance effort, albeit a weak one. Collectively and within the limitations of the dataset, both the regression analysis and Kendall's tau suggest the in-degree mean can be used as an indicator (weak predictor) of the effort required to maintain a CBS in the operational phase of the system

Descriptively, Figure 6 further shows that the effort required to maintain a CBS crosses from less-than-typical to more-than-typical effort in between 3.0 and 3.5 for the in-degree mean of the system. Systems with in-degree means lower than 3.0 have lower than average maintainability scores indicating that they require less perceived effort to maintain than those systems with a high in-degree mean. Conversely, those CBS with in-degree means above 3.5 exhibited more perceived effort to maintain. While this is a relative measure, it is an indicator early in the design phase that an architect should consider other COTS products or different COTS product arrangements to design a system that is easier to maintain in the O&M phase of the program and reduce the overall lifecycle cost of the system.

6 Analysis and Discussion

From the results, it is possible to derive observations about the relationship of the perceived effort to maintain the CBS and the architectural interactions of the systems. Further insight can be gained by evaluating the in-degree model relative to the existing COTS maintainability approaches.

6.1 BASIS Technique

The BASIS technique is a three-step process to assist in COTS selection. The third step is most like the in-degree mean process; however, the BASIS technique requires significant knowledge of the COTS component's external interfaces and how those interfaces interact with glue code. The BASIS technique is not focused on the O&M phase of the program nor the effort associated with maintaining the COTS installation base. Instead, it is focused on choosing the COTS products to meet the system requirements and minimize development costs.

The in-degree mean process and the BASIS technique are complementary and, together, can be used to develop a CBS that meets the system requirements and delivers a maintainable system. The BASIS technique can be used to select the COTS components that most accurately meet the system requirements, and the in-degree mean process can assist in choosing between multiple COTS components that meet the requirements but require different levels of effort in the maintenance phase. The two approaches do not significantly overlap.

6.2 COCOTS Cost Model

The COCOTS Cost Model is used primarily as a software-costing tool; therefore, there is no significant overlap with the in-degree mean model. COCOTS is focused on the effort to interface custom software with the external interfaces of the COTS products including the glue code that is often used to link multiple COTS products together. While the model is planned to extend into the maintenance phase, the work is

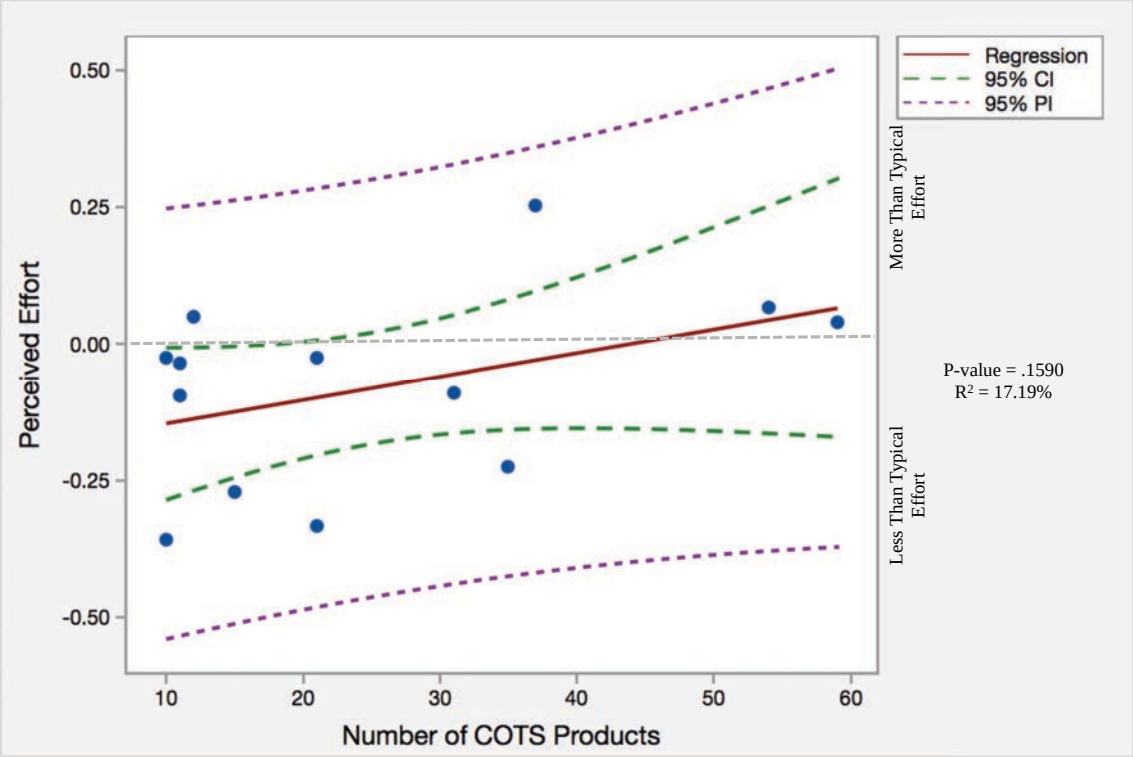

Figure 7: Plot demonstrating relationship between perceived effort and the number of COTS products in a CBS.

not yet complete; therefore, the in-degree process adds to the areas currently covered by the COCOTS Cost Model.

6.3 COTS-LIMO

The COTS-LIMO model attempts to identity a break-even point where maintenance costs increase disproportionately to the number of COTS products in the system. The theory is that there is some number of COTS components beyond which the system is simply unmaintainable because the effort associated with that number of COTS products is too great. This study tested the COTS-LIMO on the CBS dataset.

Figure 7 shows the comparison of the perceived effort (normalized) to the system size (number of COTS products). The plot suggests the expected form of relationship between the number of COTS and the perceived effort to maintain the CBS, which aligns with intuitions about COTS maintainability. But this relationship is not statistically significant (p-value = .1590) nor is the level of explained variability high ($R^2 = 17.2\%$).

From the empirical data, it may be inferred that the number of COTS in a CBS is only a portion of the story. The data show that the number of interactions between the COTS components has a stronger influence on the effort to maintain the CBS than just the number of COTS in the system.

6.4 Maintenance Delta

The theory of the Maintenance Delta asserts that there is a relationship between the Power-law and the number of architectural interactions in a CBS. Specifically, the theory states that the difference in the Cumulative Distribution Function (CDF) for the system and the CDF of a Power-Law distribution indicates the maintainability of the system [6]. For each of the systems under evaluation, the Kolmogorov-Smirnov

goodness-of-fit test [65] demonstrated that the system does not follow a power-law distribution. The α from equation (1) associated with each system is less than 2, and typical power-law distributions have $2 \leq \alpha \leq 3$ [28]. So, the systems in the current sample set do not follow a power-law distribution. It is possible that the sizes of the CBS under evaluation are simply too small to demonstrate power-law characteristics. However, for systems with a single operating system, the hub of the node (the operating system) is connected to every other node. This one node in a small system does not allow the system to follow the power-law curve: it causes α to be less than 2 and beyond the typical range of the power-law interval.

6.5 Discussion

The results support certain observations. First, there are some products that have almost as many dependencies as the operating system: Java is one of these products. Because of the high number of dependencies, the operating system and those products with almost as many dependencies as the operating system must be treated with special care when planning maintenance. For instance, careful planning must occur when upgrading products with many dependencies simply because of the testing that is required ensuring that all architectural interactions are satisfied. It has been the lead author's experience that Java is treated as a product that can be updated with little planning or coordination. The data suggest that the number of architectural interactions involved with Java and similar products requires that either they be upgraded only when complete system testing can be performed or that mitigation strategies be implemented to reduce the number of architectural interactions with these individual products during the design phase to make the CBS maintenance activities require less effort.

Second, the data suggest that system architects rarely have a comprehensive list of COTS products that comprise the system; therefore, the understanding of the architectural interactions in the system is incomplete. For example, in the data section for System T3, the architect only listed seven COTS products in the response to the request for a list of COTS products that comprise the system. But, as all of the architectural interactions were mapped, an additional five COTS products were discovered in the underlying dependencies. System T3 was not unusual in this finding. Every other system had at least two additional underlying COTS products that were not identified in the original request. Without a complete understanding of the COTS products in the system, it is more difficult to maintain the system. Additionally, without understanding the way the COTS products interact with each other, maintenance activities associated with the COTS products may have unintended consequences which may result in system outages or additional effort to properly manage changes to the system. The data also suggest that personnel with the knowledge required to apply the knowledge are often not available in the later phases of the development lifecycle.

Third, basing the CBS on a single operating system generally reduces the overall maintenance effort. Systems A, F, K, M1, M2, M3, T2 and T3 are all based on a single operating system, and the average perceived effort for these systems is 3.7. The systems with more than one operating system, including a hypervisor as an operating system, have an average in-degree mean of 5.0. While this observation is intuitive, even CBS based on a single operating system can exhibit high levels of perceived maintenance effort. Systems F and T3 score high on both the in-degree mean and the average perceived effort even though they are based on a single operating system. So, the selection of COTS products and the CONOPS of the system still must be considered even when utilizing a single operating system.

7 Conclusions

7.1 Conclusions

The in-degree mean process offers a straightforward way for architects, integrators and maintainers to determine the maintainability of a COTS deployment based on information available early in the development phase of a CBS. The model is intuitive as maintainers of the system expect that the more interactions between COTS products, the more effort changing those products will require. The maintainability of the

COTS deployment in a CBS is an area that has received little attention, so the approach offers a way to understand the complexity of the deployment through design structure matrices and network analysis. The combination of these well-known tools provides a framework to compute the in-degree mean, a measure of maintenance effort for a CBS. The in-degree mean can be used to compare deployments for competing architectures or simply as a measure of the maintainability of the CBS architecture.

The perceived effort associated with maintaining 13 systems was compared to the in-degree mean of the CBS derived from the architectural interactions of the systems. The comparison yielded a practical, intuitive and statistically-significant result indicating that the in-degree mean does correlate to the effort to maintain the systems.

The measure is useful in the beginning of the design phase to determine the maintainability of the COTS installation. The artifacts derived from the model are useful throughout the CBS lifecycle to assist in maintenance activities and to determine the level of effort required to maintain the COTS product installation base. As the system matures, the architectural dependencies of the COTS installation can be updated to increase the fidelity of the model.

7.2 Contributions

This work makes two separate contributions to the body of knowledge around CBS – the in-degree mean model and a clearer definition of design attributes that contribute to complexity in the COTS installation of a CBS. The in-degree mean model builds on existing models and gives additional insight into the maintainability of a CBS in the O&M phase of a project. No other CBS metrics or models currently consider this phase of the lifecycle when assessing effort associated with the development of a CBS, and the most developed models are only focused on the effort to maintain the custom code that is written to be delivered along with CBS instead of the effort to maintain the COTS-based installation. Because the O&M phase of a project is the longest and most expensive portion of the system's lifecycle, considering the maintainability of the CBS during this phase can help lower the overall lifecycle cost of the system. Additionally, the in-degree mean model enables architects to create systems that require less effort to maintain which should increase the success rate of the development and deployment of CBS.

Second, the work adds the system CONOPS to the list of design attributes that add complexity to the COTS installation. Previous research identified dependencies that COTS products create themselves through their installation requirements; however, system architects create additional dependencies in the COTS installation in the way data flows are designed in the system and the way information is stored in the CBS. Including the system CONOPS in the design characteristics that add to complexity creates a more complete understanding of the dependencies between COTS products and enables the system architect, integrators and maintainers to make more informed decisions on the maintenance activities associated with the CBS.

7.3 Practical Implications and Application

The in-degree mean is a straightforward calculation that gives the architect, system integrator and system maintainer knowledge of the dependencies between the COTS products in a CBS. With this information, many decisions can be made related to the design and maintenance of the CBS.

The in-degree model is also applicable to Free and Open Source Software (FOSS), Government Off-the-Shelf Software (GOTS) and Research Off-the-Shelf (ROTS) Software. The only difference between these software products and COTS is the method of procurement. The other attributes of COTS software are the same and the dependencies between these types of software products and COTS are identical. For this research, FOSS and COTS were treated identically and combined as a single product type. As an example, Figure 4 shows Java, a FOSS product, included with other software dependencies. Including FOSS and other types of software products is important as they are increasingly used to build modern CBS.

System architects can use the in-degree model to determine if the COTS installation for the project can be maintained within a typical range of effort. Because the model only requires the system CONOPS and

the installation manuals of the COTS products, the architect can compute the in-degree mean early in the design phase and make COTS product decisions or CONOPS changes to decrease the effort to maintain the system. In a system proposal, the architect can use the model to predict the level of effort required to maintain the overall system design in the maintenance phase with a view to lowering project bid metrics. Or, the architect can use the in-degree mean to demonstrate to a customer that one design is more robust than another design assuming information about both systems is available.

System integrators can use the artifacts derived during the creation of the in-degree model to focus effort on the COTS products that create the most complexity in the design. The DSM is a tool with decades of research demonstrating its applicability in managing system complexity [56,66]. Because all architectural interactions are mapped into a DSM in the creation of the in-degree model, the system integrators can use the DSM to determine which products have the most dependencies and then employ SMEs on those products to reduce the risk to the COTS installation.

Having a better understanding of the maintainability of the CBS allows the system maintainer to fine-tune project bids associated with effort to maintain the system. A low in-degree mean indicates that a system maintainer can lower the effort in a bid to maintain the system. Similarly, a high in-degree mean indicates that the system maintainer should bid a high level of effort to maintain the system. Higher fidelity bidding models decrease the financial risk to the system maintainer and assist in determining the appropriate level of support for the O&M phase of the project.

7.4 Limitations / Topics for Further Study

The systems in this study ranged from ten to 59 COTS products and were gathered from three separate and unrelated sources. While the sizes in the sample set are typical, larger systems exist. For example, two additional systems with more than 150 COTS products were excluded from the analysis because no one with current knowledge of the system was available to assist with understanding the architecture and assessing maintenance effort. Future work should include these "super systems" to determine if the in-degree mean applies to the very large systems.

The current approach combines multiple systems with various lengths of time in the operational phase of the program– some over 13 years. It seems reasonable that these systems have been updated over their lifecycle to lower the effort associated with maintaining the system. If one area of the system required significant effort to maintain early in the project's operational phase, changes could have been made to decrease the effort and create a more maintainable system later in the operational phase. The current survey approach does not account for these changes; therefore, the length of time a system has been in the operational state may influence the perceived effort to maintain the system.

Management of the deployments of a single version of a CBS to multiple locations will contribute to the overall effort to maintain the system. The research method identifies the activities associated with the deployment to multiple sites as a factor that contributes to perceived effort; however, many of the systems in the dataset did not feature multiple deployments. Future work should more thoroughly consider the effort associated with deploying to multiple locations.

The current model is focused only on the complexity associated with the COTS installation in the system. A CBS has the potential to have other dependencies including those on hardware components, custom code components and other external interfaces. Future work should expand to include other potential dependencies and the impact of those dependencies on the effort required to maintain the system.

The measurement of maintenance effort for CBS is an area that also warrants further study. For example, future research could consider direct tracking of labor hours associated with maintaining the COTS portion of a system. Identifying the actual COTS maintenance effort would eliminate the subjectivity introduced by surveying an expert on each individual system. Alternatively, if perceived effort is retained as a measure of CBS maintainability, the survey questionnaire could be expanded, the scoring approach to "N/A" responses could be refined, and the weighting factors used to establish the maintainability index could be evaluated.

Finally, the current sample set only included systems that were maintained by the same organization

that developed the system. One system in the original request was maintained by an organization that did not develop the system, and because of this unique characteristic, it was excluded from the evaluated sample set. Many contracting offices are moving to a model where one company develops a system and hands the complete system to another organization to maintain and enhance. Therefore, future work should include investigations into the applicability of the model in this unique maintenance paradigm.

Contributions: This paper was produced from M.S.'s dissertation. W.L and A.E. was the co-advisor of M.S' dissertation committee. Both co-advisors review and edited the paper. J. S. helped for statistical analysis using statistical analysis package "R"

Funding: This research received external funding from Raytheon Company through Transdisciplinary Ph.D. Program on "design, Process, and Systems" at TTU.

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] Kiss, J., Kosztyan, Z. (2009). The importance of logic planning in case of IT and innovation projects. Applied Studies in Agribusiness and Commerce. pp. 15.20.

[2] Riaz, M., Mendes,E., and Tempero, E. (2009). A systematic review of software maintainability prediction and metrics. In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM '09). IEEE Computer Society, Washington, DC, USA, 2009, pp. 367-377. DOI=http://dx.doi.org/10.1109/ESEM.2009.5314233

[3] Loukides, M. (2014). Revisiting "What is DevOps,"OReilly Radar. (http://radar.oreilly.com/2014/06/revisiting-what-is-devops.html)

[4] Smeds J., Nybom K., Porres I. (2015). DevOps: A Definition and perceived adoption impediments. in: Lassenius C., DingsØyr T., Paasivaara M. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2015. Lecture Notes in Business Information Processing, vol 212. Springer, Cham.

[5] Abts, C. (2002). COTS-Based systems (CBS) functional density – A heuristic for better CBS design. In Proceedings of the First International Conference on COTS-Based Software Systems (ICCBSS '02), John C. Dean and Andrée Gravel (Eds.)., Springer-Verlag, Berlin, Heidelberg, 1-9.

[6] Clark, B. (2007). Added sources of costs in maintaining COTS intensive systems. Crosstalk. 2007, Vol. 20 issue 6 pp. 4-8.

[7] Mitchell, C. (2009). Quality in interdisciplinary and transdisciplinary postgraduate research and its supervision: ideas for good practice. Institute for Sustainable Futures, University of Technology, Sydney (UTS) prepared for the ALTC Fellowship, Zen and the Art of Transdisciplinary Postgraduate Research, Sydney.

[8] M. Mitchell. (2009). *Complexity: A Guided Tour*. Oxford University Press, Inc., New York, NY, USA.

[9] Badampudi, D., Wohlin, C., Petersen, K. (2016). Software component decision-making: In-house, OSS, COTS or outsourcing- A systematic literature review. *Journal of Systems and Software,* Vol. 121 pp. 105-124.

[10] Vale, T., Crnkovic, I., de Almeida, E. S., Neto, P. (2016). Twenty-eight years of component-based software engineering, *Journal of Systems and Software,* Vol: 111, pp. 128-148.

[11] Yanes, N., Sassi, S. B. Ghezala, H. (2017). Ontology-based recommender system for COTS components. *Journal of Systems and Software*, Vol: 132, Page: 283-297.

[12] Abts, C., Boehm, B., Bailey Clark, B. (2000). COCOTS: a COTS software integration cost model, European Software Control and Metric Conference, Munich, Germany, pp. 325-333.

[13] Närman, P., Sommestad, T., Sandgren, S., Ekstedt, M. (2009). A framework for assessing the cost of it investments. Proceedings of Portland International Conference on Management of Engineering Technology, PICMET August 2009, pp. 3154-3166.

[14] Vigder, M. , Kark, A. (2006). Maintaining COTS-based systems: start with the design, Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems (ICCBSS'05), Orlando, Florida, Feb 2006.

[15] Boehm, B., Abts, C. (1999). Cots integration: plug and pray? *IEEE Comput.,* Vol. 32, no. 1, pp. 135-138.

[16] Morisio, M. Seaman,C., Basili, V., Parra, A. (2002). COTS-based software development: Processes and open issues, *Journal of Systems and Software*, Vol. 61, Issue: 3, pp. 189-199.

[17] Smith J. D., Hybertson D. (2002). Implementing large-scale COTS reengineering within the United States Department of Defense. In: Dean J., Gravel A. (eds) COTS-Based Software Systems. ICCBSS 2002. Lecture Notes in Computer Science, vol 2255. Pp 245-255 Springer, Berlin, Heidelberg.

[18] Rosa, W., Packard,T., Krupanand, A., Bilbro, J. W., Hodal, M. M. (2013). COTS integration and estimation for ERP, *The Journal of Systems & Software,* 86, pp. 538550.

[19] Basili, V. R., Boehm, B. W. (2001). COTS-based systems top 10 list. *Computer,* Vol. 34, no. 5, pp. 91-95.

[20] Davis, L., Gamble, R. F., Payton, J.(2002). The impact of component architectures on interoperability. *Journal of Systems and Software*, Vol. 61, issue 1, pp. 31-45.

[21] Kozlov, D., Koskinen, J., Sakkinen, M., Markkula, J. (2008). Assessing maintainability change over multiple software releases. *Journal of Software Maintenance and Evolution-Research and Practice*, Vol. 20, pp. 3158.

[22] Ballurio K., Scalzo B., Rose L. (2002). Risk Reduction in COTS Software Selection with BASIS. In: Dean J., Gravel A. (eds) COTS-Based Software Systems. ICCBSS 2002. Lecture Notes in Computer Science, 2002, vol 2255. Springer, Berlin, Heidelberg.

[23] Clark, B., Cook, D., Lowery, H., Stults, K., Nilsen, K. (2007). COTS Integration, Crosstalk: *The Journal of Defense Software Engineering*, Vol 20, No. 6, pp 4-24.

[24] Smith, M. W. Tate, D., Lawson, W. D. (2013). Measuring the Maintainability of Commercial Off-the-Shelf (COTS) Based System Deployments: A Network-Based Approach, ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2013, V005T06A039V005T06A039.

[25] Anderson, R. B. (2001). The power law as an emergent property. *Memory & Cognition*, Vol. 29, pp. 10611068, doi:10.3758/BF03195767.

[26] Barabási, Albert-László and Eric Bobabeau (2003). Scale-free networks. *Scientific American*, Vol. 288, No. 5, pp. 6069. JSTOR, www.jstor.org/stable/26060284.

[27] Ted G. Lewis, (2009). *Network Science: Theory and Applications*. Wiley Publishing.

[28] Clauset, A., Shalizi, C. (2009). Power-law distributions in empirical data, *SIAM Rev.*, vol. 51, no. 4, pp. 661703.

[29] Ardimento P, Bruno G, Caivano D, and Visaggio G. (2007). A maintenance oriented framework for software components characterization, 11th European Conference on Software Maintenance and Reengineering, p 10.

[30] Allen, E. B., Gottipati, S. & Govindarajan, R. (2007). Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach.*Software Qual J*, 15, pp. 179212, doi:10.1007/s11219-006-9010-3

[31] Davis, J., LeBlanc, R., (1988). A study of the applicability of complexity measures. *IEEE Trans. Software Eng.*, Vol. 14, pp. 1366-1372.

[32] Gao, S and Li, C.(2009). Complex network model for software system and complexity measurement. WRI World Congress on Computer Science and Information Engineering, Vol. 7, pp. 624-628.

[33] Harrison, W. (1992). An entropy-based measure of software complexity. *IEEE Transactions on Software Engineering*, Vol. 18, pp 1025-1029.

[34] Salman, N. (2006). Complexity metrics AS predictors of maintainability and integrability of software components. *Journal of Arts and Sciences*, pp. 39-50.

[35] Siddhi, P., Rajpoot, V. K. (2012). A Cost estimation of maintenance phase for component based software.*IOSR Journal of Computer Engineering*, (IOSRJCE), Vol. 1, pp.150.

[36] BhutaJ. and Boehm, B. (2007). A Framework for Identification and Resolution of Interoperability Mismatches in COTS-Based Systems. In Proceedings of the Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS '07). IEEE Computer Society, Washington, DC, USA, 2007, 2-. DOI=http://dx.doi.org/10.1109/IWICSS.2007.1

[37] Garlan, D., Allen, R., Ockerbloom, J. (1995). Architectural Mismatch or Why it's hard to build systems out of existing parts. Proceedings 17th International Conference on Software Engineering, April 1995.

[38] Abts, C. and Boehm, B. (1997). COTS Software Integration Cost Modeling Study. USCCSE tech. Report 98-520, USC Center for Software Engineering, Los Angeles, CA.

[39] Lewis T. G. (2009). *Network Science: Theory and Applications.* Wiley Publishing.

[40] Suh, N. (2006). Application of axiomatic design to engineering collaboration and negotiation. in 4th International Conference on Axiomatic Design, Firenze, 2006.

[41] Klimenko, A. Y.(2014). Complexity and intransitivity in technological development. *J. Syst. Sci. Syst. Eng.*, Vol. 23, no. 2, pp. 128152.

[42] Zheng W. (2003). Entropy, information, noise - studies on system evolution. *Journal of Systems Science and Systems Engineering*, Vol. 12, pp.2-12.

[43] Yassine, A., Whitney, D. , Daleiden, S., & Lavine, J.(2003). Connectivity maps: Modeling and analysing relationships in product development processes. *Journal of Engineering Design,* Vol 14, no. 3, pp. 377-394, DOI: 10.1080/0954482031000091103.

[44] Collins, S. T., Yassine, A., S. Borgatti, P. (2009). Evaluating product development systems using network analysis. *Syst. Eng.,*, Vol. 12, pp. 55-68.

[45] Sosa, M. , Eppinger, S., Rowles, C. A (2007). Network approach to define modularity of components in complex products. *Journal of Mechanical Design*, Vol. 129, no. 11, pp. 1118-1129.

[46] Doyle, J., Alderson, D., Li, L., Low, S. (2005). The robust yet fragile nature of the Internet. Proceedings of the National Academy of Sciences of the United States of America.

[47] Shirinivas, S. G., Vetrivel, S., Elango, N. M. (2010). Applications of graph theory in computer science: An overview. *International Journal Eng. Sci.,* Vol. 2, no. 9, pp. 4610-4621.

[48] Newman, M.(2002). Assortative mixing in networks. *Physical Review Letters,* Vol 89, no. 20, 208701.

[49] Saaty, T. L. (2004). Fundamentals of the analytic network process – Dependence and feedback in decision-making with a single network. *J. Syst. Sci. Syst. Eng.*, Vol 13, pp. 129157.

[50] Schneidewind, N., Hinchey, M. (2009). A complexity reliability model. 20th Intl. Symp. Software Reliability Eng., pp. 1-10.

[51] Newman, M. (2010). *Networks: An introduction.* Books.Google.com., Oxford University Press.

[52] Brandes, U., Lerner, J., Lubbers, M., McCarty, C., Molina, J. (2008). Visual statistics for collections of clustered graphs. Proc. of the IEEE Pacific Visualization Symposium, pp. 47-54.

[53] Newman, M. E. J. (2003). The structure and function of complex networks. *SIAM Rev.*, Vol. 45, no. 2, pp. 167256.

[54] Smith, M. W.(2015). Measuring the maintainability of commercial off-the-shelf (COTS) based systems: A complexity approach (Doctoral dissertation).

[55] Hayes,J. H., Zhao, L. (2005). Maintainability prediction: a regression analysis of measures of evolving systems. *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 601604.

[56] Eppinger,S. D. (2001). Innovation at the speed of information. in *Harvard Bus. Rev.*, Vol. 79, pp. 149-158.

[57] Bhuta, J., Mattmann, C.A., Medvidovic, N., Boehm, B. (2007). A Framework for the assessment and selection of software components and connectors in COTS-based architectures. *Software Architecture,* 2007. WICSA '07. The Working IEEE/IFIP Conference. on, pp. 6-6.

[58] Bertoa, M., Troya, J., Vallecillo, A. (2006). Measuring the usability of software components. *Journal of Systems and Software*, Vol 79; Issue 3, pp. 427-439.

[59] Babbie, Earl R. (1989). *The Practice of Social Research.* Belmont, CA, Wadsworth.

[60] Correia, J. P., Kanellopoulos, Y., Visser, J. (2009). A survey-based study of the mapping of system properties to iso/iec 9126 maintainability characteristics. Software Maintenance 2009. ICSM 2009. IEEE International Conference, pp. 61-70.

[61] Gliem, J. A., Gliem,R. R. (2003). Calculating, interpreting, and reporting cronbach's alpha reliability coefficient for likert-type scales. Midwest R Midwest Research to Practice Conference in Adult, Continuing, and Community Education.

[62] Bailey, K. D. (2008). *Methods of Social Research.* Simon and Schuster. com.

[63] Stark, R., Roberts, L. (20020. *Contemporary Social Research Methods.* Belmont, CA, Wadsworth/Thompson Learning.

[64] Kurtz, N. (1999). *Statistical analysis for the social sciences.* Boston: Allyn and Bacon.

[65] Haldar, A., Mahadevan, S. (2000). *Probability, reliability and statistical methods in engineering design.* John Wiley & Sons, Inc. New York.

[66] Eppinger S. D., Whitney, D. E., Smith, R. P., Gebala, D. A. (1994). A model-based method for organizing tasks in product development. *Res Eng Design*, Vol. 6, no. 1, pp. 1–13.

About the Authors

Dr. Michael Smith is a Principal Engineering Fellow at Raytheon in Richardson, Texas. Michael has more than 23 years experience designing, integrating and maintaining COTS-based systems both domestically and internationally. He is currently the Technical Director for Infrastructure where he is focused on optimizing the design and deployment of platforms for large complex CBS.

Dr. W. Lawson's research and creative activities encompass both technical and interdisciplinary domains. Technical research has focused on geotechnical engineering for transportation applications, primarily in the areas of soil/structure interaction, roadway maintenance, unsaturated/expansive soils, and foundations. Interdisciplinary research has focused on engineering education, engineering ethics, engineering professionalism, assessment, and the efficacy of distance learning methods. He has also published on geotechnical engineering history and is actively interested in topics such as engineering judgment, risk, and decision making.

Dr. A. Ertas, Professor of Mechanical Engineering and director of the Academy for Transdisciplinary Studies at Texas Tech University, received his masters and Ph.D. from Texas A&M University. He had 12 years of industrial experience prior to pursuing graduate studies. Dr. A. Ertas has been the driving force behind the conception and the development of the transdisciplinary model for education and research. His pioneering efforts in transdisciplinary research and education have been recognized internationally by several awards. He is a Senior Research Fellow of the ICC Institute at the University of Texas Austin, a Fellow of ASME, a Fellow of Society for Design and Process Science (SDPS), Founding Fellow of Luminary Research Institute in Taiwan, an honorary member of International Center for Transdisciplinary Research (CIRET), France, and a member of ASEE. Dr. Ertas has earned both national and international reputation in engineering design. Dr. Ertas is the author of a number of books, among them: Ertas, A. and Jones, J. C., *The Engineering Design Process,* John Wiley & Sons, Inc., first addition 1993 and second edition 1996; Ertas, A., *Prevention through Design (PtD): Transdisciplinary Process,* funded by the National Institute for Occupational Safety and Health, 2010; Ertas, A., *Engineering Mechanics and Design Applications, Transdisciplinary Engineering Fundamentals*, CRC Press, Taylor & Francis Group, 2011; A. Ertas, A., *Transdisciplinarity Engineering Design Process,* John Wiley & Sons, 2018. He has edited many research books specific to transdisciplinary engineering design, among them: Ertas (editor), *Transdisciplinarity: Bridging Natural Science, Social Science, Humanities & Engineering*, ATLAS Publications, 2011; B. Nicolescu, B. and Ertas A., (editors), *Transdisciplinary Theory and Practice*, ATLAS Publications, 2013; Nicolescu, B., Ertas, A., (Editors), *Transdisciplinary Education, Philosophy, & Applications,* ATLAS Publications, 2014; Ertas, A., Nicolescu, B., S. Gehlert, S., (Editors), *Convergence: Transdisciplinary Knowledge & Approaches to Education and Public Health*, ATLAS Publishing, 2016; Nicolescu, B., Yeh, R. T., Ertas, A., (Editors), *Being Transdisciplinary.* ATLAS Publishing, 2019. He has also edited more than 35 conference proceedings. Dr. Ertas' contributions to teaching and research have been recognized by numerous honors and awards. He has published over 175 scientific papers that cover many engineering technical fields. He has been PI or Co-PI on over 40 funded research projects. Under his supervision more than 180 MS and Ph.D. graduate students have received degrees.

Dr. James G. Surles received B.S. degrees in Mathematics and Computer Science from McNeese State University in 1995 and an M.S. and Ph.D. in Statistics from the University of South Carolina in 1997 and 1999, respectively. Dr. Surles came to Texas Tech University in 1999, where he is currently full Professor. His main research interests are Reliability and the Exponentiated Weibull and Burr type X lifetime models, but he also enjoys working with researchers from around Texas Tech on a variety of research projects.