# Systems Engineering in a COTS World

Richard L. Sorensen
Vitech Corporation
2070 Chain Bridge Road, Suite 100
Vienna, Virginia 22182

**Abstract.** Using traditional systems engineering practices to accommodate commercial off-the-shelf (COTS) components requires focusing on those steps within the design process that allow identification of the constraints imposed by COTS items early in the design process. Two traditional systems engineering processes are reviewed for their insights into the major tasks, the goals, and the dependencies. With this understanding, the focus moves to discussion of a proposed approach for applying systems engineering practices using a process that retains a requirements-driven methodology while simultaneously adapting to the boundaries imposed by COTS items, thereby converging on a solution.

## Introduction

The growing availability of commercial off-the-shelf (COTS) components to satisfy design solutions in many technical areas brings benefits and challenges to the systems engineering discipline. COTS components offer significant benefits in four principal areas:

- Cost benefit of large scale production [R&D and tooling costs are spread over a larger market segment]
- Industry-sponsored efforts toward standardization of interfaces
- Ready availability
- Time-to-market

Hand-in-hand with these benefits comes some challenges. Most significant to the systems engineer, specification of candidate COTS components bound the solution space. Where previously the design solution was often bounded only by physics, manufacturing capability, and customer requirements, now we have to map requirements into a constrained solution space. This bounding of the solution space requires reverse engineering of identified COTS components during the traditional top-down systems engineering process (Long 2000).

A particular concern with COTS is the all-too-natural tendency to bypass the requirements analysis process and move straight into a design solution. Drivers such as time-to-market and customers with unclear requirements make it tempting to bypass the requirements process but introduce significant risk. Specifically, in a world where feature-rich COTS components are available, there is a very real danger of over-engineering a solution. This introduces the potential for increased costs to implement and support the solution, as well as adding functionality that may not be needed nor desired. More importantly, there is a risk of failing to identify and capture customer requirements in the delivered solution.

This paper looks at two common systems engineering processes, top down and reverse engineering, for the features applicable to engineering a COTS-driven design. Following that, a suggested approach for COTS systems engineering is presented followed by an example.

## Systems Engineering Methodologies in Review

We can draw on two forms of systems engineering processes to help guide the development of a COTS-driven design: the traditional top down process and the reverse engineering process.

**Top Down Process.** Top down systems engineering processes, depicted in Figure 1, emerged out of an environment where the end result was a new device/component/system built to specifications derived from originating requirements. The resulting design emerges from a process that starts with requirements [steps 0 & 1], captures system boundaries and architecture constraints [steps 2 & 3], develops functional behavior from those requirements [steps 4 & 5], derives component hierarchy from the behavior and allocates behavior to those components [steps 6 & 7], defines necessary interfaces and finally allocated those behaviors to components to produce component specifications, manufacturing directions, etc to build these components. This process has an inherent feature that bounds the end product only by the requirements implemented in the systems engineering process.

The top down process sets the form for requirements analysis, regardless of the application. The methodology of defining and refining requirements, developing functional behavior models, and developing component and system architectures to encompass these behaviors is a process we need to carry forward into our COTS-driven design effort.

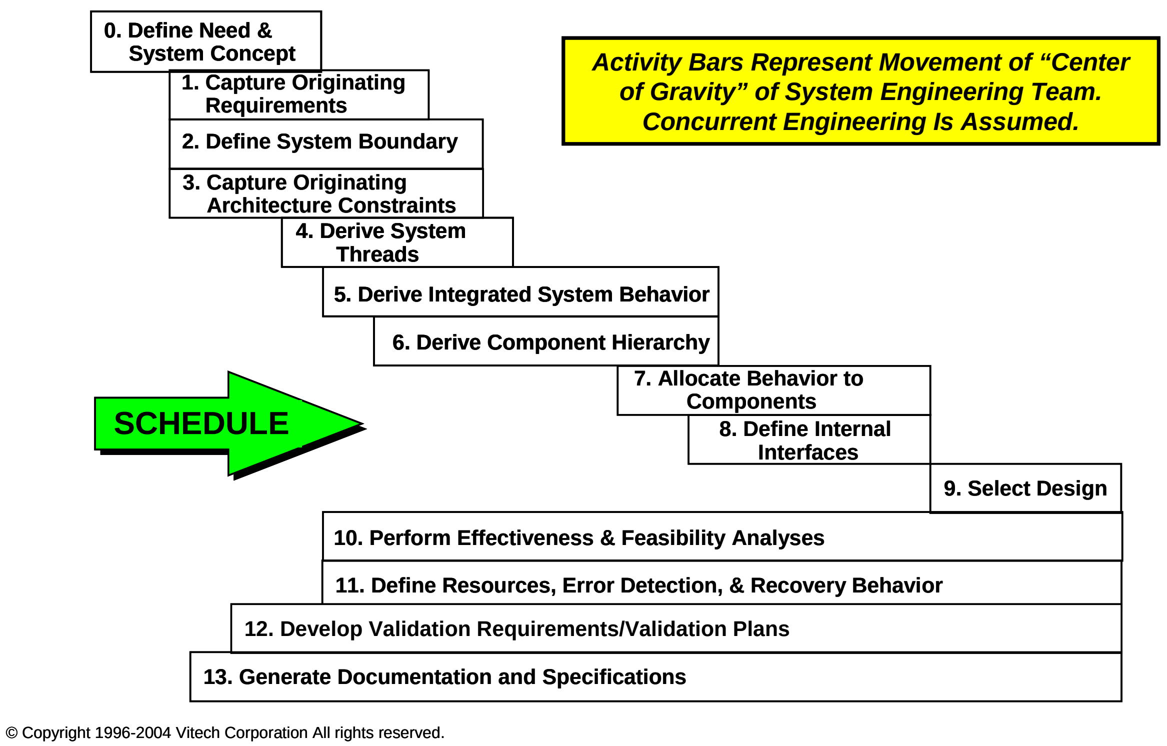
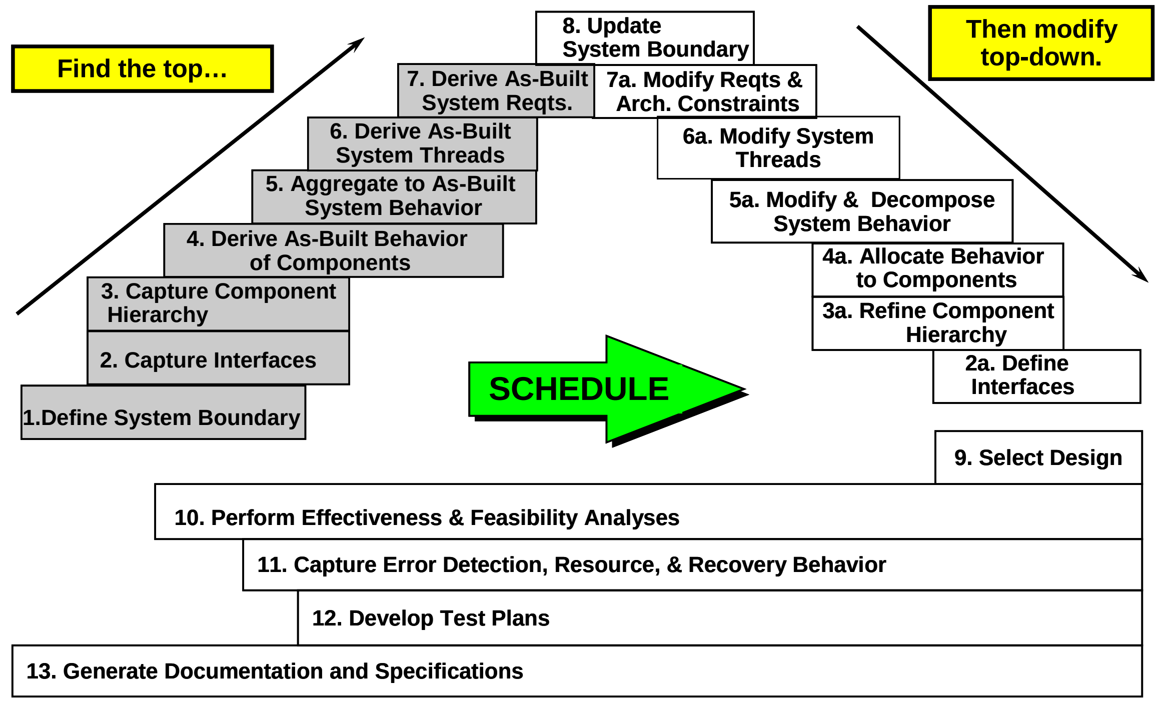

0. Define Need & System Concept
1. Capture Originating Requirements
2. Define System Boundary
3. Capture Originating Architecture Constraints
4. Derive System Threads
5. Derive Integrated System Behavior
6. Derive Component Hierarchy
7. Allocate Behavior to Components
8. Define Internal Interfaces
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
11. Define Resources, Error Detection, & Recovery Behavior
12. Develop Validation Requirements/Validation Plans
13. Generate Documentation and Specifications
Activity Bars Represent Movement of "Center of Gravity" of System Engineering Team. Concurrent Engineering Is Assumed.
SCHEDULE
© Copyright 1996-2004 Vitech Corporation All rights reserved.

Figure 1. Top Down Systems Engineering Process

**Reverse Engineering.** Reverse engineering processes, illustrated in Figure 2, emerged out of a need to discover and document system behavior and implied requirements for in-place solutions that needed to be modified and/or replaced. The process documents existing system boundaries, interfaces, and components [steps 1, 2, 3], derives existing behavior of components and the system [steps 4, 5, 6], and identifies a requirements set that reflects the as-built system.

Once this effort is complete, modifications are applied [if required] to reflect new requirements [steps 8, 7a], system and component behavior is updated to accommodate the changes [steps 6a, 5a], behaviors allocated to components [steps 4a, 3a], interfaces defined [step 2a], and the design selected [step 9]

In this form, the resulting solution is constrained by the in-place behavior of the existing system. One significant aspect of this process, applicable to the systems engineering process applying COTS components, is the idea of modelling existing physical components given their inherent functional behavior and interfaces to find the boundaries they impose onto the solution space.



Find the top…
Then modify top-down.
8. Update System Boundary
7. Derive As-Built System Reqts.
7a. Modify Reqts & Arch. Constraints
6. Derive As-Built System Threads
6a. Modify System Threads
5. Aggregate to As-Built System Behavior
5a. Modify & Decompose System Behavior
4. Derive As-Built Behavior of Components
4a. Allocate Behavior to Components
3. Capture Component Hierarchy
3a. Refine Component Hierarchy
2. Capture Interfaces
2a. Define Interfaces
1.Define System Boundary
SCHEDULE
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
11. Capture Error Detection, Resource, & Recovery Behavior
12. Develop Test Plans
13. Generate Documentation and Specifications

**Figure 2. Reverse Engineering Process**

## Systems Engineering COTS-Driven Design

The challenge COTS presents to the systems engineer is to use these systems engineering processes in an environment where the solution space is bounded by the existing functional and physical aspects of the COTS components. This is both a design process and an integration process. The significant aspect introduced by COTS is that the design process is now constrained by a set of pre-existing components, which introduce functionality that may or may not be required by a specific design solution.

**Requirements Process.** The approach to systems engineering practices that invoke a COTS-based solution is to address requirements—both originating requirements and an initial decomposition—to quantify expected behaviors and performance and how they may constrain a solution. Figure 3 shows the initial steps taken in a COTS-driven design. Using an integrated systems engineering design environment such as Vitech Corporation's CORE®, the tools are readily available to capture source documents, extract the originating requirements, and then decompose these into lower level requirements so that there is solid traceability backwards and forwards as progress is made toward a design. It is not uncommon to find that originating documents also expose some expected performance parameters and constraining features. Performance parameters and constraints may need some decomposition at this point, again with the goal of quantifying expected behavior of the solution.

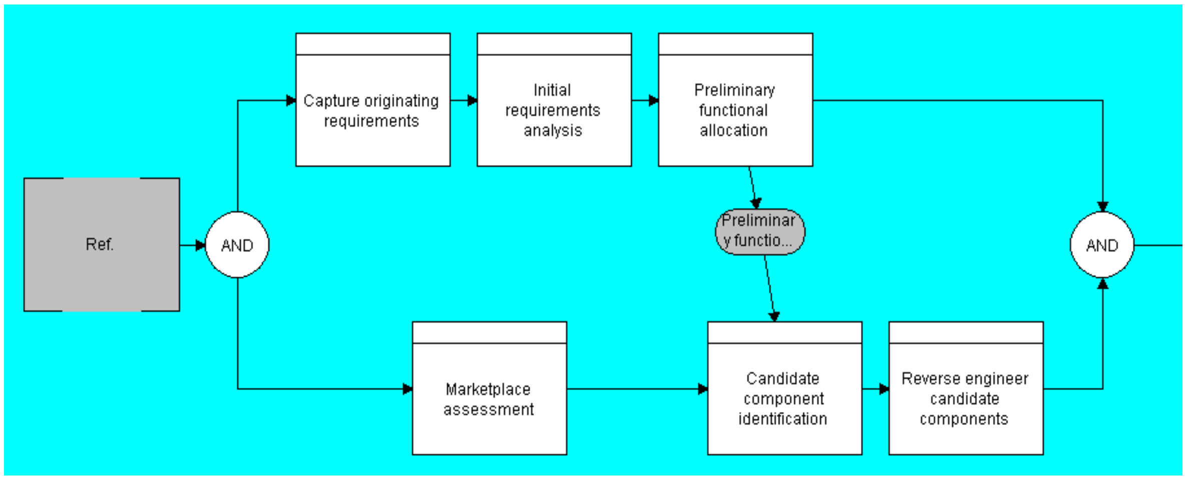

Ref.
AND
Capture originating requirements
Initial requirements analysis
Preliminary functional allocation
Preliminar y functio...
AND
Marketplace assessment
Candidate component identification
Reverse engineer candidate components

**Figure 3. Initial Steps in COTS-Driven Design**

**Reverse Engineering of Candidate Components.** In parallel with the traditional requirements capture and decomposition, a COTS-driven design process explores the marketplace for available COTS components and identifies those applicable to the solution space presented by the requirements. Those candidate components identified are then explored using the techniques from the reverse engineering process to derive functional behavior(s) and interfaces. In the example shown in Figure 4, the physical components of a Cisco PIX515 Firewall are exposed and partially decomposed. This process is repeated for each candidate component identified to expose the functions provided and subsequently map required functions to those provided by the COTS components.

**Functional Modeling of Requirements.** The next step is to map requirements into functional behavior models, applying any known constraints. Domain expertise is especially important in this step to guide the functional behavior model development along the lines of existing industry functional allocation to components. An example of this is the computer networking industry which typically has allocated common behaviors to devices called routers, switches, hubs, etc. While individual vendors will supply devices that have functionality unique to their branding, in general, devices such as a router or switch can be expected to perform common functions regardless of the vendor.

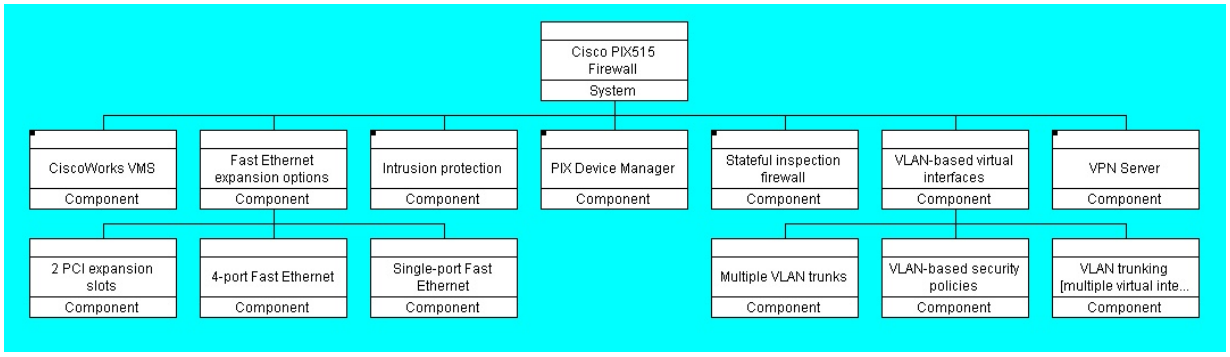


**Figure 4. Example of Reverse Engineered COTS System Decomposition**

**Solution Space Analysis. O**nce generic functional behaviors are identified, the next step is to identify candidate COTS hardware/software components based upon this high-level assessment. Often, trade studies are warranted to search the component marketplace to identify candidate components, taking into account customer preferences, purchasing restrictions, etc. It is not unusual to have a customer who has an existing purchasing relationship with a given vendor or to have a customer with a specific vendor preference due to in-place knowledge [i.e. customer X already has network operations staff trained in vendor Y's management software and prefers not to introduce new training requirements into their existing operation]. It is also a possibility that purchasing restrictions prevent a customer from buying equipment from certain sources regardless of the cost or operating benefit. These conditions must be applied early in the selection of components to prevent wasted effort analyzing components that are not viable candidates in a design solution. They should also be identified as constraints within the system design repository.

The brand-unique functionality must be examined in each case to determine what unique functions are supplied, how they may prove advantageous [or not] to a given design solution, and their cost effectiveness in a given solution space. This step involves reverse-engineering the candidate components into their functional behavior(s) and assessing the goodness of fit to the desired functional behavior. These steps will often iterate as candidate components are selected and subsequently rejected due to a poor fit or the availability of better solutions. This iterative process is shown in Figure 5.

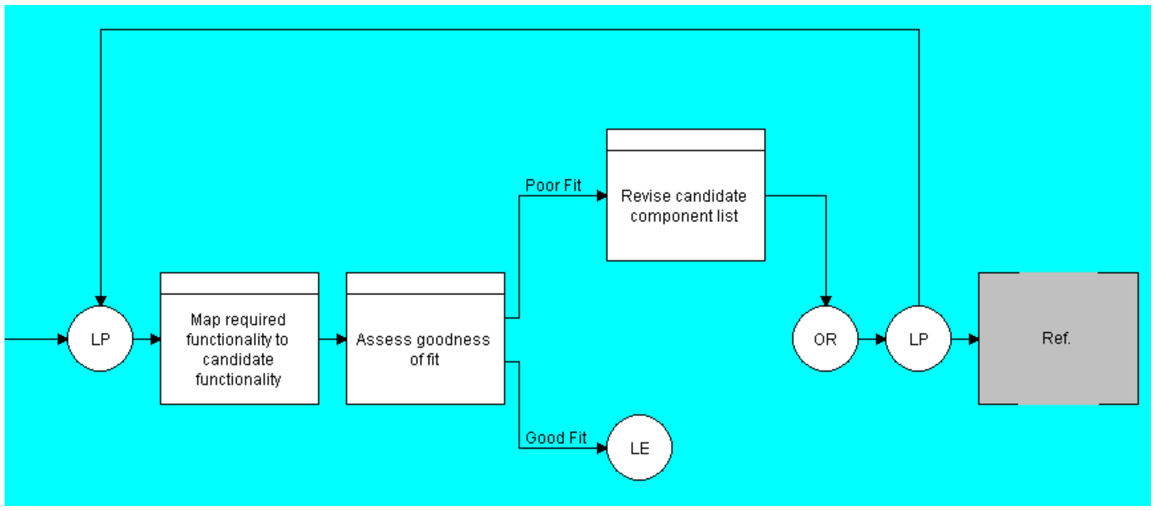

LP
Map required functionality to candidate functionality
Assess goodness of fit
Poor Fit
Revise candidate component list
Good Fit
LE
OR
LP
Ref.

**Figure 5. Solution Space Analysis Process**

**Impacts of Excess Functionality.** The existence of excess functionality—those functions that exist in COTS components that exceed the functionality required by customer requirements—poses some unique challenges. Whenever excess functionality exists in a design solution, we need to look at the impacts to reliability and operations. One fundamental idea from reliability engineering applies: added components impact reliability in a negative way. Each component and each component's functions introduce another potential point of failure. The fewer components that are introduced in the final design, the better! Operationally, the presence of these unused functions poses a risk. When one of these excess functions fails, it introduces a potential nuisance factor if in-place fault detection capabilities report the failure. Additionally, what do we do when it fails? Does the user replace an otherwise functioning component, incurring unnecessary cost? Does the user leave the component in-place and risk losing visibility to other failures?

In addition, if the added capability is desired but not required by the customer, it may be appropriate to change the scope of the project to document a change in the requirements. Another option is to document the existence of these added capabilities and begin planning for a future expansion, retaining the scope of the existing project, schedule, and budget.

**Other COTS Considerations.** Systems engineers must address many logistics issues introduced by the presence of COTS components in a delivered system. When logistics pipelines are critical, the issue of having little or no control over internal configuration suggests that component servicing and repair may be necessary at the black box level, introducing a potentially higher life cycle cost. Another potential approach is replacement on an equivalent unit basis rather than an identical unit basis as vendor-supported service lifetimes for COTS components tend to be shorter, driven by the marketplace and emerging technologies that cause rapid obsolescence. In the example that follows, key COTS components that were current when the design was produced and procured in 2001 were declared 'End of Sale, End of Life' by late 2003.

Addressing the specific issue of lack of configuration control of COTS components, (Long 2000) presented some options to consider:

- Embed sufficient capability in the design to accommodate future changes to the COTS components
- Freeze the design with one configuration of the COTS component(s) [with the inherent demand on the logistics community to procure and maintain sufficient replacement components for the life of the system]
- Plan to periodically refresh the COTS content of the system through system update cycles

Each option needs careful consideration of the implications and exploration with the customer to determine what the customer can and will support.

The introduction of short life cycles for COTS components, coupled with the presence of excess functionality, introduces another point for potential problems. The COTS marketplace is often driven by the presence of features that allow a potential customer to discriminate between vendor A's products and vendor B's products. Marketplace drivers influence these vendors to continually introduce new features in their products to set themselves apart. Older products are sold at discounts, new products replace them on the shelves, and when it is time to replace or re-procure a COTS component, it is an equivalent, but updated, component that is available. The new features are seen as excess functionality initially, but present the opportunity for incorporation into the system after deployment. This after-the-fact incorporation can modify the fundamental behavior of the system and introduce undesired and unintended performance.

**COTS Impact on Systems Engineering Process.** The impacts of COTS-driven design on the top down systems engineering process are highlighted in Figure 6. Steps 0 thru 5 reflect a tradition requirements capture and derivation process. Steps 6, 7, & 8 reflect the reverse-engineering aspects of capturing COTS behavior and interfaces. Note that these steps are an expansion of the activities of step 3, Capture Originating Architecture Constraints, added to give focus to specific activities needed by the introduction of COTS in the design process. Steps 4a, 5a, and 6a reflect the integration phase, bringing the necessary COTS components together in a coherent picture driven by the original requirements for the system being designed. The design selected in step 9 is a result of the confluence of customer requirements and the functional behaviors supplied by selected COTS components.

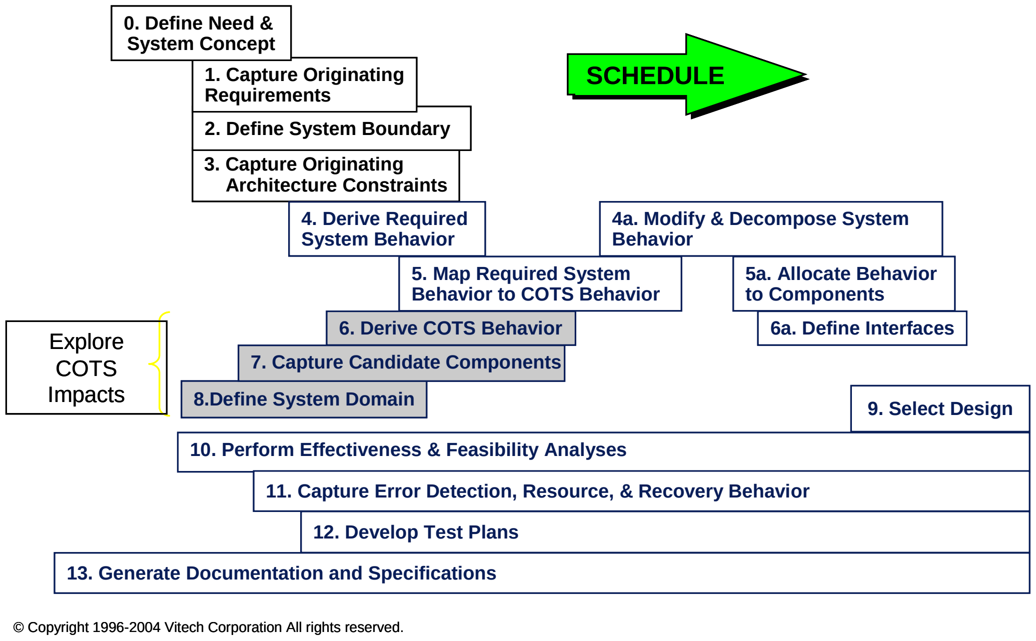

0. Define Need & System Concept
1. Capture Originating Requirements
2. Define System Boundary
3. Capture Originating Architecture Constraints
SCHEDULE
4. Derive Required System Behavior
4a. Modify & Decompose System Behavior
5. Map Required System Behavior to COTS Behavior
5a. Allocate Behavior to Components
6. Derive COTS Behavior
6a. Define Interfaces
Explore COTS Impacts
7. Capture Candidate Components
8.Define System Domain
9. Select Design
10. Perform Effectiveness & Feasibility Analyses
11. Capture Error Detection, Resource, & Recovery Behavior
12. Develop Test Plans
13. Generate Documentation and Specifications
© Copyright 1996-2004 Vitech Corporation All rights reserved.

**Figure 6. COTS-Driven Design Impacts on Top-Down Systems Engineering Process**

## Example Application of COTS Process

This section describes how systems engineering process were used for an actual COTS-driven design project. A customer wanted to integrate their legacy models and simulations and provide the integrated capabilities as assets to their entire enterprise. The goals included capturing embedded knowledge about how their systems worked, facilitating knowledge transfer to a new generation of design engineers, improving the response times to requests for analysis products, and setting the stage for future growth of the business segment through an enhanced design process they expected to emerge following initial implementation. The Concept of Operations document identified two high-level requirements as shown in Figure 7.

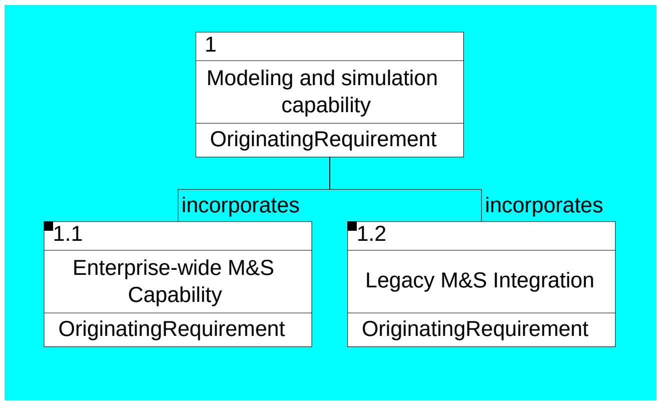


**Figure 7. Originating Requirements**

These enterprise-wide modeling and simulation (M&S) requirements were decomposed in discussions and trade studies to further refine the needs and expectations of the users and implementers. These decompositions are shown in Figures 8 and 9.

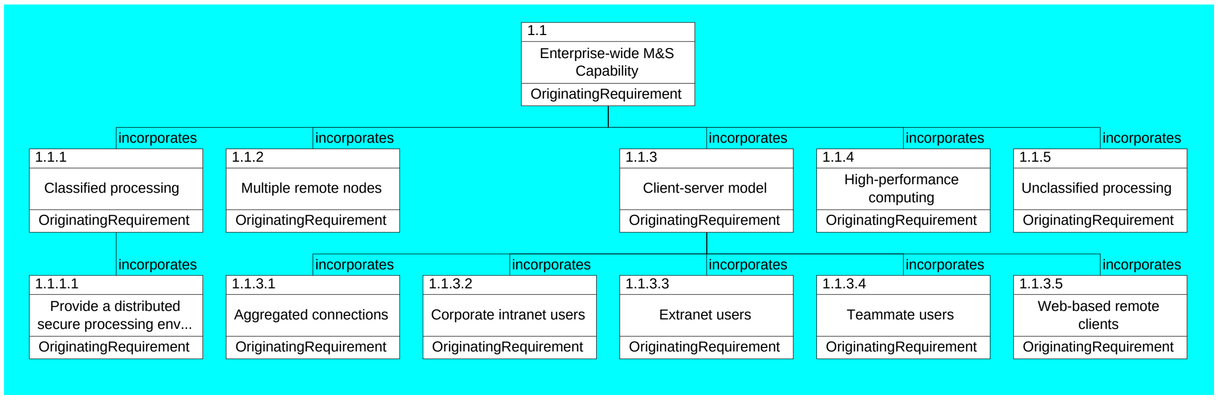


**Figure 8. Enterprise-wide M&S Capability Requirements**

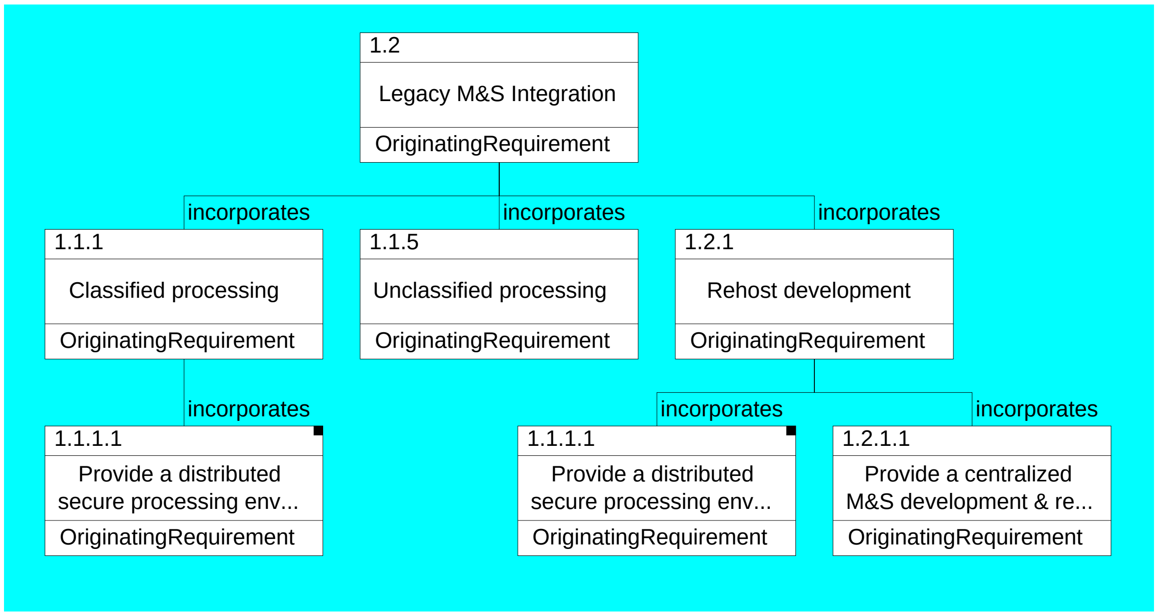


**Figure 9. Legacy Systems Requirements**

With this level of decomposition, a number of requirements were beginning to emerge. First was an enterprise-wide accessibility supporting:

- Distributed clients on an existing extranet,
- Clients on existing corporate intranets,
- Clients accessing through aggregated connections through firewalls, and
- External clients with web-based access.

The solution also required the capability to operate in both classified and unclassified modes.

Secondly, the customer needed the capability to centralize the rehost efforts of legacy codes. The rehost process also invoked the requirement to operate in both classified and unclassified modes.

Further research and requirements analysis were necessary to understand the implications of existing corporate intranets and the in-place extranet, available connectivity, and the associated restrictions which accompanied added connections. Of particular importance was understanding the implications of the secure processing requirement. Government documents were identified that further defined the requirements for a secure processing environment and these were added to the derived requirements and factored into the COTS-driven design.

**Initial Functional Analysis.** Experienced network design engineers and security advisors were consulted to map the resulting requirements into an initial functional behavior. Figure 10 describes the top level functional behavior of one node of the M&S capability, with functions related closely to typical networking components. This was decomposed further to expose lower level functions with added detail until a complete shared understanding of required capability existed. Figure 11 shows a further partial decomposition of required functions for protecting the network being designed.

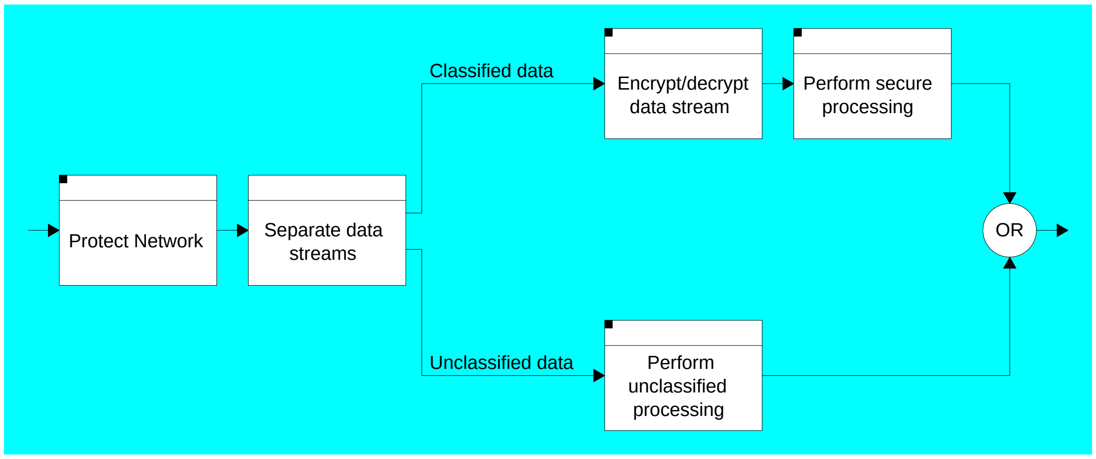

Classified data
Encrypt/decrypt data stream
Perform secure processing
Protect Network
Separate data streams
OR
Unclassified data
Perform unclassified processing

**Figure 10. Top Level Functional Behavior Model – One Node**

What emerged was a set of functions that accommodated both unclassified and classified processing environments. Internal aggregation functions were invoked to support multiple development workstations connecting to servers and to each other to implement a distributed processing capability.

External functions necessary to meet basic network traffic routing and network access security requirements were achieved by implementing IP address mapping and delivery [routing], IP address qualification and packet filtering [firewall functions], and encryption/decryption functions.

**Solution Space Analysis.** The customer had indicated a strong preference for Cisco components early in the project, so the candidate COTS component list was restricted to Cisco components. Cisco 3640 and 3660 routers were identified to meet the IP address mapping and portioning requirements, a Cisco PIX515 Firewall was identified to meet IP address qualification and packet filtering requirements, Cisco 3524 switches were identified to meet the port aggregation requirements, and an encryptor was identified to meet the security requirements.

The next step was mapping required functional behavior to physical components. Figure 12 shows an example of the traceability hierarchy diagram where the network protection functions have been mapped to the PIX515 firewall components. This mapping shows functions allocated via the performs relation to components of the previously reverse-engineered PIX515.

Examining the excess functionality of this initial allocation of hardware components revealed that the PIX515 firewall could meet the required IP address mapping requirements due to the existence of an upstream router and minimal IP routing required within the design solution. The VLAN Trunking capability of the PIX515 supplanted the router functions and was implemented in the design.

This resulted in a reallocation of functions that removed the Cisco 3640 router, simplifying the design and lowering acquisition and operations and maintenance (O&M) costs. The modified network interconnect diagram is illustrated in Figure 13.

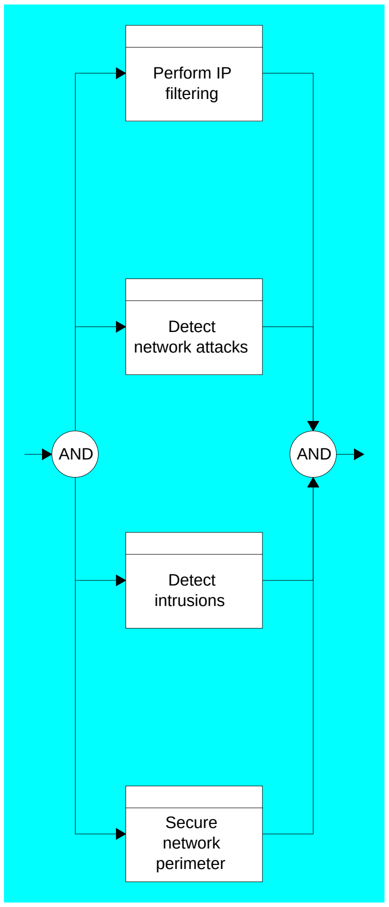


**Figure 11. Network Protection**

At this point, all required hardware functionality had been accommodated and the project proceeded to the implementation phase. During installation and checkout, it was determined that the interface requirements of the encryptor had not been fully documented, a frequent problem in the COTS arena. To implement the configuration requirements of the encryptor, an external switch needed to be incorporated upstream on the outbound data path. An examination of the excess functionality available in the in-place hardware identified a virtual LAN (VLAN) capability delivered by the Cisco 3524 switch, along with the necessary ports that had been put in place for future expansion capacity. The VLAN feature was implemented, dividing the 24 ports on the switch into two VLAN's and creating a 2 port virtual switch to accommodate the encryptor. The resulting final network interconnection diagram is shown in Figure 14.

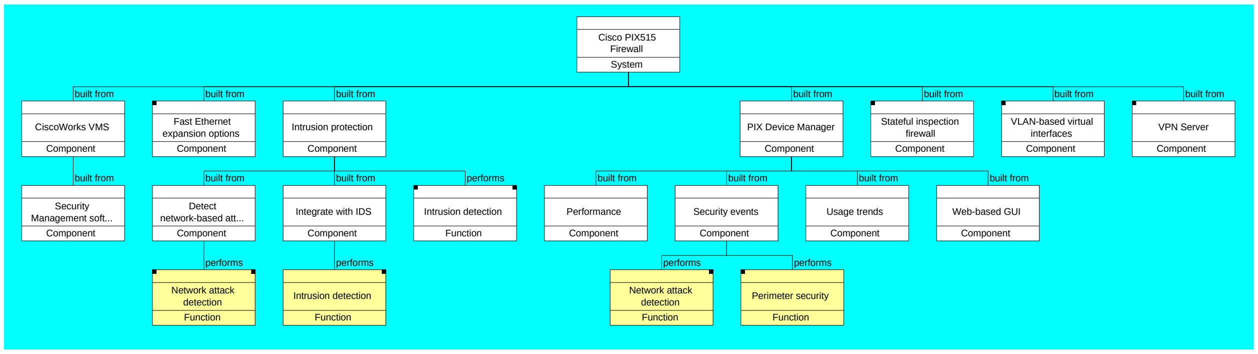


**Figure 12. System Traceability Hierarchy**

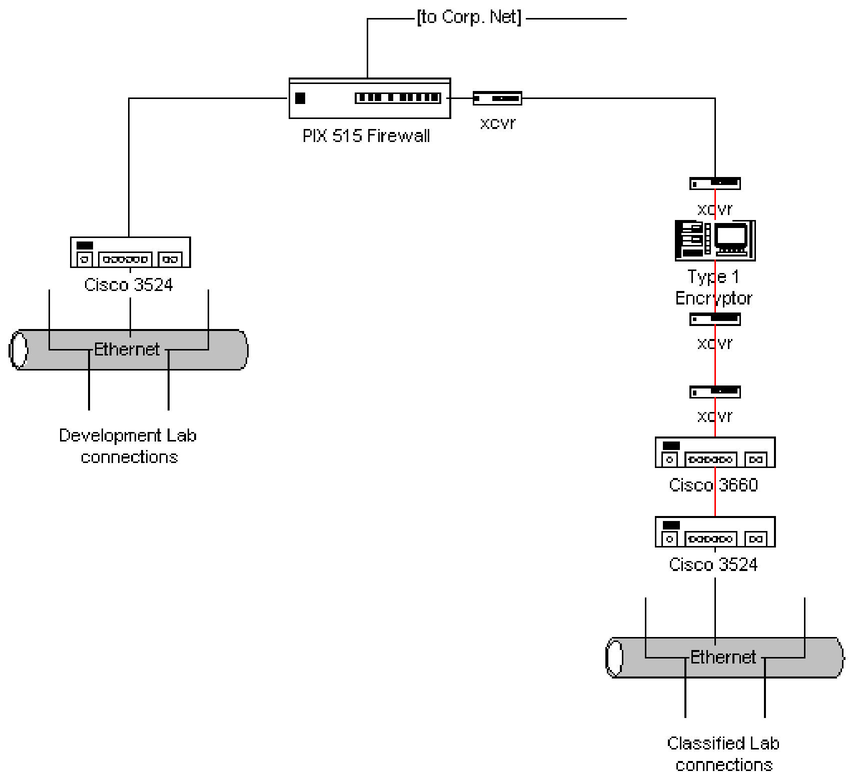


**Figure 13. Updated Network Interconnection**

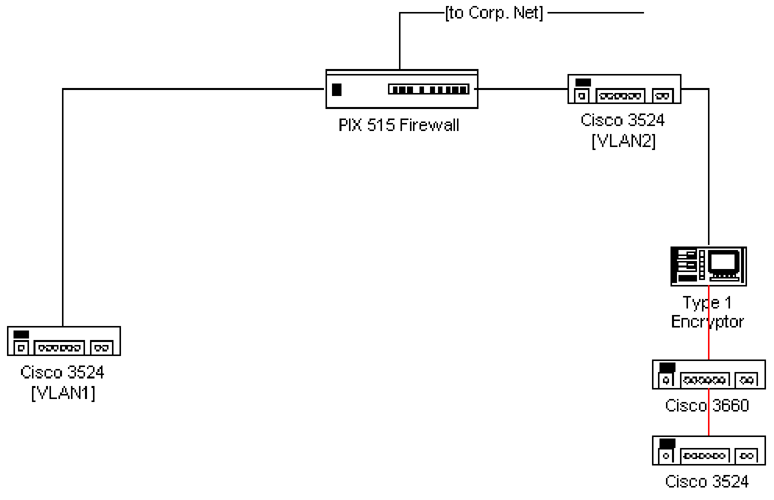


**Figure 14. Final Network Interconnection**

**Lessons Learned.** In this particular project, having excess functionality resulted in project success. However, the lack of complete knowledge of the interfaces presented by the encryptor presented the implementation team with significant challenges and resulted in many weeks of delays until the complete set of requirements were obtained from the COTS vendor. Had the detailed interface requirements been known, the design process would have been able to accommodate them and implementation would have proceeded on schedule.

Beyond the cost of additional engineering hours, the impact of this problem and the subsequent solution was to implement a previously excess functionality. This changed the configuration of the designed solution. In itself this would not normally be an issue, however, one year after initial implementation the customer requested a duplicate capability be procured and installed at another facility. If the change that was made to facilitate the encryptor interface had not been captured, it could have easily resulted in problems when the hardware bill of materials was reviewed. A further complication resulted when Cisco retired the 3524 Switch series. It was no longer possible to purchase identical equipment, and not all switches offer the VLAN capability implemented to support the encryptor.

## Summary

The ability to quickly deliver cost-effective solutions through the use of COTS hardware/software components can be a customer-imposed restraint that faces many systems engineers in today’s business environments. Having a process that guides COTS implementation to meet customer goals is crucial. However, understanding how to apply systems engineering processes such as top down and reverse engineering to a COTS-driven design is crucial to meet customer goals.

Systems engineers involved in the COTS application process need to be aware of the unique challenges presented by the constraints imposed by the components they choose as well as the risks of over-engineering any solution. Using the systems engineering process to guide the analysis of COTS delivered functionality and mapping that functionality to required functionality helps to expose potential problems as well as potential growth areas.

## References

Long, James E., “COTS: What You Get (In Addition to the Potential Development Savings).” *Proceedings of the Tenth Annual International Symposium of the International Council on Systems Engineering* (Minneapolis, MN, July 16-20, 2000). INCOSE, Seattle, WA, pp 609-611.

## Biography

Richard L. Sorensen has over twenty-three years of experience in hands-on system operations and maintenance as well as modeling and simulation, information systems architecture development and integration, business process reengineering, database design, hardware and software systems integration, and enterprise architecture planning. As a senior systems engineer with Vitech Corporation, he brings his experience to bear on numerous customer projects.

Prior to joining Vitech Corporation, he was the systems architect for multiple special purpose integration labs and a systems engineer supporting a number of medical and software-intensive systems at TRW. During employment with Rockwell International, he developed system test requirements for a system test program and supported the development of the associated test sets and test procedures as well as development of command and control system simulations. While at Martin Marietta Aerospace, he developed a weapon system simulation to verify fault detection/fault isolation methods, system failure information, and command and control system interaction. Mr. Sorensen holds a Bachelors of Science in Computer Engineering from Syracuse University. He has completed extensive graduate coursework in computer engineering, systems engineering, and modeling & simulation.