U. S. Department
of Transportation
**Federal Railroad
Administration**

Commercial-Off-The-Shelf (COTS) Hardware and Software for Train Control Applications: System Safety Considerations

DOT/FRA/ORD-03/14

**Final Report
April 2003**

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2003	Final Report October 2001

4. TITLE AND SUBTITLE
Commercial Off-The-Shelf (COTS) Hardware and Software for Train Control Applications: System Safety Considerations

5. FUNDING NUMBERS
R2115/RR293

6. AUTHOR(S)
Anne Clough

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Charles Stark Draper Laboratory, Inc.*
555 Technology Square
Cambridge, MA 02139

8. PERFORMING ORGANIZATION REPORT NUMBER
DOT-VNTSC-FRA-02-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
*Under contract to:
U.S. Department of Transportation
Research and Special Programs Administration
Volpe National Transportation Systems Center
55 Broadway
Cambridge, MA 02142-1093

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
DOT/FRA/ORD-03/14

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161. This document is also available on the FRA web site at www.fra.dot.gov.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The objective of this study was to assess the feasibility of using commercial off-the-shelf (COTS) processor-based systems for safety-related railroad applications. From the safety perspective, the fundamental challenges of using COTS products are most pronounced within the product evaluation and safety assurance phases of the traditional development life cycle. Even in the case of COTS components that can be considered commodities, such as certain microprocessors and operating systems, there are some necessary life cycle modifications. First, more iteration at the system level is required to accommodate component evaluation and selection as well as deal with technical and business requirements and tradeoffs. Second, the entire development life cycle has to be modified to address realities such as component evaluation and the high pace of change in the commercial market of today. To address these realities, an iterative life cycle was proposed for systems incorporating COTS components. Finally, traditional safety verification and validation methodologies were contrasted with safety validation approaches for integrating COTS components into hardware and software systems. This effort resulted in a structure to support the development of a system safety program plan for COTS operating system platforms executing safety-critical applications.

14. SUBJECT TERMS
commercial off-the-shelf (COTS), operating system, positive train control, modified off-the-shelf (MOTS)

15. NUMBER OF PAGES
44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

PREFACE

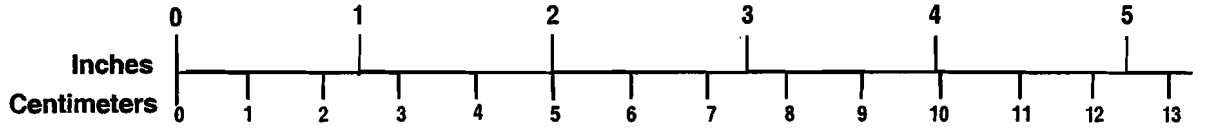
The Federal Railroad Administration (FRA) is responsible for ensuring the safety of locomotive equipment in the United States. A primary concern of the FRA is the proper use of computer technology in the implementation of safety-critical functions, such as locomotive onboard control systems in newer high-speed systems, as well as in conventional rail systems. This report describes the development of a methodology designed to ensure that a sufficiently high level of safety is achieved and maintained in computer-based systems that employ commercial off-the-shelf technology. Adequate safety is necessary whether the systems are used in new applications or are used to replace or enhance existing equipment.

This report was prepared by Volpe National Transportation Systems Center and Charles Stark Draper Laboratory personnel. We would like to acknowledge the FRA sponsor, Steve Ditmeyer, Director, Office of Research and Development, for providing technical guidance.

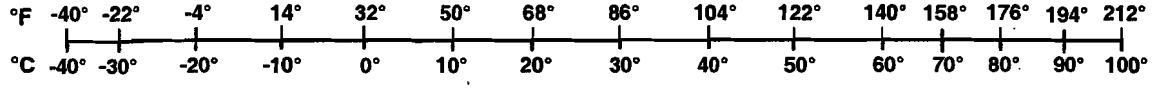
METRIC/ENGLISH CONVERSION FACTORS

ENGLISH TO METRIC	METRIC TO ENGLISH
LENGTH (APPROXIMATE)	**LENGTH (APPROXIMATE)**
1 inch (in) = 2.5 centimeters (cm)	1 millimeter (mm) = 0.04 inch (in)
1 foot (ft) = 30 centimeters (cm)	1 centimeter (cm) = 0.4 inch (in)
1 yard (yd) = 0.9 meter (m)	1 meter (m) = 3.3 feet (ft)
1 mile (mi) = 1.6 kilometers (km)	1 meter (m) = 1.1 yards (yd)
	1 kilometer (km) = 0.6 mile (mi)
AREA (APPROXIMATE)	**AREA (APPROXIMATE)**
1 square inch (sq in, in^2) = 6.5 square centimeters (cm^2)	1 square centimeter (cm^2) = 0.16 square inch (sq in, in^2)
1 square foot (sq ft, ft^2) = 0.09 square meter (m^2)	1 square meter (m^2) = 1.2 square yards (sq yd, yd^2)
1 square yard (sq yd, yd^2) = 0.8 square meter (m^2)	1 square kilometer (km^2) = 0.4 square mile (sq mi, mi^2)
1 square mile (sq mi, mi^2) = 2.6 square kilometers (km^2)	10,000 square meters (m^2) = 1 hectare (ha) = 2.5 acres
1 acre = 0.4 hectare (he) = 4,000 square meters (m^2)	
MASS - WEIGHT (APPROXIMATE)	**MASS - WEIGHT (APPROXIMATE)**
1 ounce (oz) = 28 grams (gm)	1 gram (gm) = 0.036 ounce (oz)
1 pound (lb) = 0.45 kilogram (kg)	1 kilogram (kg) = 2.2 pounds (lb)
1 short ton = 2,000 pounds (lb) = 0.9 tonne (t)	1 tonne (t) = 1,000 kilograms (kg) = 1.1 short tons
VOLUME (APPROXIMATE)	**VOLUME (APPROXIMATE)**
1 teaspoon (tsp) = 5 milliliters (ml)	1 milliliter (ml) = 0.03 fluid ounce (fl oz)
1 tablespoon (tbsp) = 15 milliliters (ml)	1 liter (l) = 2.1 pints (pt)
1 fluid ounce (fl oz) = 30 milliliters (ml)	1 liter (l) = 1.06 quarts (qt)
1 cup (c) = 0.24 liter (l)	1 liter (l) = 0.26 gallon (gal)
1 pint (pt) = 0.47 liter (l)	
1 quart (qt) = 0.96 liter (l)	
1 gallon (gal) = 3.8 liters (l)	
1 cubic foot (cu ft, ft^3) = 0.03 cubic meter (m^3)	1 cubic meter (m^3) = 36 cubic feet (cu ft, ft^3)
1 cubic yard (cu yd, yd^3) = 0.76 cubic meter (m^3)	1 cubic meter (m^3) = 1.3 cubic yards (cu yd, yd^3)
TEMPERATURE (EXACT)	**TEMPERATURE (EXACT)**
[(x-32)(5/9)] °F = y °C	[(9/5) y + 32] °C = x °F

QUICK INCH - CENTIMETER LENGTH CONVERSION



QUICK FAHRENHEIT - CELSIUS TEMPERATURE CONVERSION



For more exact and or other conversion factors, see NIST Miscellaneous Publication 286, Units of Weights and Measures. Price $2.50 SD Catalog No. C13 10286

Updated 6/17/98

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

LIST OF FIGURES

LIST OF ACRONYMS

COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-The-Shelf
DCOM	Distributed Common Object Model
FRA	Federal Railroad Administration
GPOS	General Purpose Operating System
GUI	Graphical User Interface
I/O	Input/Output
IEEE	Institute of Electrical and Electronics Engineers
MOTS	Modified Off-The-Shelf
ORB	Object Request Broker
PC	Personal Computer
PHA	Preliminary Hazard Analysis
PHL	Preliminary Hazard List
PTC	Positive Train Control
RSPP	Railroad Safety Program Plan
RTX	Real Time Extensions
SEI	Software Engineering Institute
SEL	Software Engineering Laboratory
V&V	Verification and Validation

EXECUTIVE SUMMARY

Consistent with the general trend of replacing custom systems with commercial off-the-shelf (COTS) hardware and software products, the Federal Railroad Administration (FRA) is interested in tracking the possibilities of incorporating COTS components into safety-relevant operational systems. While COTS components have some definite advantages over custom systems, safety and reliability considerations need to be addressed before they can be implemented.

While COTS components have been widely expected to provide significant cost savings, this has not always proved to be the case. Hidden costs include market research that must be conducted to find suitable components, extensive component evaluation and testing, licensing problems, and costs due to continual changes and upgrading of commercial components. Nevertheless, there are advantages to using commercial components, particularly in the areas where COTS components have reached the status of commodities. Microprocessors and operating systems fall in this category.

Even in the case of COTS components that can be considered commodities, there are some necessary life-cycle modifications. More iteration at the system level is required to accommodate component evaluation and selection and deal with technical, business, and requirements, tradeoffs. The entire development life cycle has to be modified to address realities, such as component evaluation, and the constant change in technology, which the use of commercial components requires throughout system development and sustainment. To address these realities, an iterative life cycle is proposed for systems incorporating COTS components.

The current safety verification and validation methodologies detailed in the FRA proposed rule for processor-based signal and train control systems (49 CFR 236 Subpart H NPRM) may require modification to address the lack of access to code when COTS components are used. A more extensive module and system verification will replace the usual method of verifications throughout development. An iterative approach is again proposed to deal with the necessity to repeat verification and validation steps when changes and upgrades to components are made.

On the operating system level, fundamental issues must be addressed at the evaluation stage of system development, including access to source code, choice of a real-time operating system versus non-real-time operating system, and operating system functionality. Once an operating system is chosen, it may even be necessary to tailor it or otherwise protect against unwanted extra functionality. Development of a software fault-tolerant architecture could provide an extra level of reliability and safety for safety-critical applications.

1 INTRODUCTION

The increasing reliability, capability, and potential cost-effectiveness of commercial off-the-shelf (COTS) products have generated significant interest in the railroad industry. The issue of paramount concern to the Federal Railroad Administration (FRA) is whether it is possible to ensure that these platforms satisfy the safety and reliability requirements for safety-critical applications like positive train control (PTC).

This report will provide insight into the advantages and disadvantages of COTS use. In addition, this report will address the fundamental differences in the way safety-critical systems incorporating COTS components are developed and how to establish the safety level of these systems. The report will also present ways to capitalize on the advantages of COTS components, while minimizing the disadvantages and emphasizing the use of COTS at the microprocessor and operating system levels.

1.1 SCOPE

The primary focus of this report is to assess the feasibility of using COTS in the railroad industry. Fundamental challenges, such as developmental life-cycle changes, product evaluation, and product safety, will be addressed. The emphasis on these topics will be on the changes in safety verification and validation (V&V) that may be necessary when COTS components are used. The implications of using commercial operating systems and appropriate recommendations will be discussed. Issues raised when using multiple COTS components in project development (i.e., more complex integration issues requiring extensive bridging software and component-based architectures) are topics for future study.

1.2 PURPOSE

The purpose of this report is to discuss the use of COTS as it relates to components on the microprocessor and operating system levels. Reliability concerns will be emphasized, and approaches for ensuring the safety of integrated COTS hardware and software systems will be detailed. Traditional safety V&V methodologies will be contrasted with safety validation approaches that can be used when COTS components are integrated into hardware and software systems. A structure will be provided to support the development of a system safety program plan for COTS operating system platforms for safety-critical applications.

1.3 DEFINITIONS

1.3.1 COTS SOFTWARE

As used in this report, a precise definition of the term COTS software is "a product sold, leased, or licensed to the general public by a commercial supplier in multiple identical copies. It is expected to be used *without modifications* by the user, with support from the supplier who retains the intellectual rights" [1].

The "without modification" distinction is emphasized due to the misconception that a COTS product can be modified by the vendor or third-party developer to meet requirements and still be considered COTS. When a COTS product is modified, many of the advantages are lost. If the vendor modifies the COTS component, the acquisition cost will increase, affecting the price advantage of using the COTS product. The modified version probably will not track updated product releases (unless the license agreement stipulates that the vendor incorporate all changes into subsequent COTS releases), and may no longer integrate easily with other COTS products. If the application developer performs the modification, the process still will be costly. The source code will need to be purchased and considerable investment time will be required to understand it. In most cases, the vendor's maintenance and design documentation will be inadequate. In either case, the modified COTS product is difficult to track to the vendor's new COTS releases. These components, known as modified off-the-shelf (MOTS), will stagnate as other products they depend on change and evolve.

Making proprietary modifications to COTS components is inconsistent with the original motivation for employing COTS. The modifier is responsible for the maintenance of the product with all its attendant costs. If modification cannot be avoided, a project will be successful only if there is a clear understanding of the potential risk incurred by modification. Therefore, system life-cycle costs of any COTS product modification must be taken into account as part of architectural and product selection decision-making.

1.3.2 VERIFICATION AND VALIDATION

Many practitioners use the terms "verification" and "validation" differently. The use of the terms verification and validation found in FRA documents adhere to the following Institute of Electrical and Electronics Engineers (IEEE) definitions (IEEE Std. 610.12-1990):

Verification: The process of determining whether or not the products of a given phase of the software development life cycle fulfill the requirements established during the previous phase.

Validation: The process of evaluating software at the end of the software development process to ensure compliance with software requirements.

1.4 COTS-RELATED ISSUES

COTS software has the potential to lower development costs, provide system flexibility, and improve the process of developing software systems. It has been touted as a way to build software systems "better, cheaper, faster."

The use of COTS components introduces new trade-offs and issues, especially with risk management, component integration, system reliability, and cost of sustainment. Component-based software engineering must be part of a large-scale paradigm shift, and represents significant retooling for most organizations.

This section will provide an overview of the advantages and disadvantages of COTS products. Any time COTS components are considered, it is necessary to deal with specific COTS-related

issues. The most important of these are summarized in the following subsections. Later sections will address the development life-cycle changes necessary to address these issues.

1.4.1 Costs

A number of costs are sometimes hidden, so organizations must conduct market research and product analyses to find suitable COTS products and alternatives. In safety-critical or security-critical applications with high integrity requirements, the cost of assessment may very quickly outweigh benefits. Securing licenses and warranties is time-consuming and costly, especially if the warranty available to the general public is not acceptable. Another cost driver is that "plug and play" functionality has not been developed for software. Integration is not always straightforward and may require the development of additional software. When multiple COTS components are integrated into a system, complex integration issues occur. These are often resolved through extensive bridging software and component-based architectures. Often, it is desirable to add software to ensure that safety issues are handled properly, or make modifications to the COTS interface to make dangerous features inaccessible. Finally, just keeping track of upgrades, to say nothing of the cost of recovery when a vendor discontinues a product or goes out of business, does much to temper the promise of cost savings [2].

Using COTS software may not actually save money, nor may savings be immediately realized or the key benefit. There is a lack of sufficient data to permit a definitive general conclusion about the cost of using COTS. However, using COTS often generates additional system capabilities and improvements for the same investment as custom development. When the price tag for custom development outstrips the available budget, employing COTS components may be the only way to field a product. With COTS, an additional advantage is the ability to implement the latest commercial technology so that, after 20 years, a fielded system can still be state-of-the-art.

A case also can be made for cost savings (and reliability) using COTS in the areas where COTS components have reached the status of commodities. Some software functions, such as databases, math functions, and probably operating systems, are so well known that it is counterproductive to build one from scratch. In these areas, COTS products provide the functionality found in a custom built component with the same functionality.

1.4.2 Quality

The first issue of concern to a developer considering the use of COTS components is the quality of the components. If the quality is insufficient, it is clearly not feasible to choose a COTS-based approach.

A COTS-based development life-cycle must incorporate evaluation of component quality. There is no "Consumer Report" for COTS software and it is unlikely that there will be in the near term for two reasons: the inherent variability of COTS products, and the complex and changing nature of the systems that use them. An initial COTS software evaluation framework has been defined [3]. Some would also like to see a verification process implemented for COTS components; however, this is not considered to be imminent [4].

1.4.3 DEVELOPMENT PROCESS CHANGES

The extent of changes incurred in the development life cycle when using COTS components should be taken into account. Whether it is the waterfall, the spiral, or other traditional life-cycle models used to develop software, there is a set of well understood activities that occur during this development. Incorporating COTS products into a system brings some pervasive changes to these development activities. Necessary life-cycle modifications that are required when using COTS are discussed in Section 2.2, and a proposed COTS development life cycle is presented in Section 2.3.

1.4.4 COTS VENDOR VIABILITY

If the system developed will be in service for a period of time, the viability of vendors becomes another crucial consideration. If a vendor discontinues a product or goes out of business, the support and maintenance for a system may disappear. It may be necessary to stipulate in the licensing agreement that, in the case of such an event, the buyer will have full rights to the source code.

1.4.5 UPGRADES AND REVISIONS

The impact of COTS upgrades on the development and maintenance cycles of the system should be taken into account. The individual user will have limited control of the frequency or content of COTS releases since the marketplace, not a user's system context, drives product changes. Upgraded functionality may not meet user needs in the time required, or may lead to a potential product and/or system mismatch. Multiple product releases may occur on varying schedules when a variety of COTS products are used.

When new products or releases emerge during development, often the old version is either not supported or compatible. While it is possible for a user to occasionally skip over a release, vendors still tend to support only a limited number of versions, and a development program that ignores a vendor's new releases cannot survive in the long term.

1.4.6 EXTRA COMPONENT FUNCTIONALITY

The potential impact of unintended functions and inadvertent activation of unused functions must be assessed in the process of determining acceptability. For more complex COTS components (like an operating system), a significant education investment is often required before the product can be fully evaluated. In cases where the product proves unsuitable, this time investment might have a zero net return [5]. (Extra component functionality is covered in more detail in Sections 2.2.4 and 5.3.1.)

1.4.7 Lack of Visibility into Source Code and System Behavior

For safety-critical applications, an overriding concern is the lack of visibility into the development process of the COTS component or access to the source code. Lack of visibility can make components hard to integrate when determining component fitness for the system or component reliability.

The most desirable situation would allow sufficient visibility and funding for execution of traditional safety V&V methodologies. Lacking this favorable situation, alternative methods for attaining confidence in the COTS products need to be defined. The challenge for system developers is to integrate COTS components into systems without compromising the strict reliability and availability requirements in critical systems.

1.4.8 Architecture Issues

The public interface to the services provided by the COTS component is important. It is necessary to assess how well the components interface with the software architecture. This will have a significant cost impact on the development of the system. The capabilities of a component must be weighed against the relative costs of integrating them into the software architecture. Regardless of the component-based architecture selected, there will be a direct impact on the availability of candidate components considered for inclusion in the system. The system architect should be prepared for the possibility of an architecture that evolves as the system is developed.

1.4.9 Business Issues

Though time consuming and costly, licenses and warranties must be secured, especially if the warranty available to the general public is not suitable. Business decisions cannot be dealt with wholly by administrative personnel, so technical personnel are needed to deal with procurement issues.

2 TRADITIONAL VERSUS COMPONENT-BASED DEVELOPMENT

2.1 FUNDAMENTAL LIFE-CYCLE CHALLENGES

The challenge in defining an appropriate life cycle for a development that includes COTS components is to address short-term and long-term life-cycle issues. In the short term, as the system is developed, the life cycle must balance system architecture with marketplace considerations and integration issues. In the long term, it is necessary to sustain and upgrade the COTS-based system. The COTS marketplace will drive continuous system evolution, requiring that systems using COTS components be engineered to accommodate marketplace imperatives. The chosen life cycle must be able to respond to vendor release and update cycles made necessary by technology change and evolution. Because of continuous change, integration will be continuous. During both development and sustainment phases, it is necessary to implement safety-related activities so that safety is not compromised.

2.2 NECESSARY LIFE-CYCLE MODIFICATIONS WHEN USING COTS

2.2.1 PARADIGM SHIFT

The essential difference in roles between the developer of a custom system and the developer of a COTS-based system has been described as "creator" versus "integrator." Essentially, the development of a custom system is an act of creation, whereas the development of a COTS-based system is an act of composition and reconciliation. This description gives the impression that the creativity has been taken out of the picture; in reality, the need to interact creatively with the system architecture has moved to the forefront. The system skills required for COTS-based development are at least as significant as those required for custom software development.

The fundamental shift from development to composition is responsible for numerous technical, organizational, management, and business changes. Some of these changes are obvious; others are subtle. If not addressed, any change can cause severe problems for the project. Organizations may need to modify their procedures and structures. In some cases, entirely new procedures may need to be created.

2.2.2 ACCOMMODATION TO CONSTANT CHANGE

COTS products will change during development and throughout system deployment. The life cycle must be flexible enough to accommodate upgrades and changes in the COTS components, which may involve the addition and/or deletion of features. Changes may occur before anything is fielded.

Because of the need to handle constant change, development and maintenance are no longer distinct [3]. Through all the changes that will occur during development and after a system is fielded, it is essential that systems maintain the required level of performance even after new components are introduced. Maintenance and redesign are not free, and sustainment costs may exceed custom development.

Hardware engineers are seeing similar problems with microprocessors and other components. Manufacturers continually develop new generations of products, thereby leading to a lifetime procurement cycle. The danger in this approach is that some systems may require redesign, so the architecture must be flexible enough to accommodate such change.

2.2.3 Requirements Analysis Differences

In COTS-based development, there should be a shifting of balance to a systems engineering point of view for technical, business, and requirements tradeoffs. In the traditional custom software development life cycle, system requirements precede software and hardware requirements definition and analysis, followed by design, coding, and testing. In COTS-based development, requirements analysis may need to be repeated a number of times before converging, and may need to be revisited during development and maintenance. This process is iterative in nature and requires simultaneous definition and tradeoffs [6].

A balance between getting the software to execute what is required by an application, and performing within the bounds of the available COTS software is a priority. Otherwise, any cost savings may quickly diminish. Looking for products with a perfect match to requirements (the normal state of the practice) is not sufficient. Instead, during the requirements analysis phase, it is necessary to have knowledge of the existing marketplace to guide in the development of functional features. Often, it is necessary to keep *a priori* requirements to a minimum and create requirements to match the market. To solicit bids for a commercial product, yet describe functional capabilities for which no commercial instances exist, would be contradictory.

The requirements also must be sufficiently generic to accommodate the differences between comparable commercial products such as Windows NT and real-time operating systems such as Windows CE [2]. A more detailed discussion of this topic is found in Section 5.2.1.

2.2.4 Evaluation and Testing of Components

Another aspect of COTS-based development not found in traditional life-cycle models is the evaluation and testing of components. Evaluation of products and technologies begins the moment the initial system idea is conceived, and forms a part of all other activities. Evaluation consists of determining the quality of a product in the context of its intended use. The evaluator must identify alternatives; define evaluation criteria in such areas as interfaces, vendor characteristics, reliability, performance, safety, and cost; and rank the alternatives against the criteria. The evaluation criteria should address several requirements including:

- function
- performance
- interface
- maintenance and support
- cost (in terms of the total life cycle)
- reliability
- safety

Perfect candidate fits are unlikely, and there always will be competing trade-offs. Since COTS components are proprietary, complete visibility into component implementation, functionality, or performance is impossible.

Activities required for demonstrating quality may be quite costly. Evaluation activities need to span the entire lifetime of a system – before a new system is designed, as a system is being implemented, and when replacing a component in an operational system. The evaluation and testing occurs when a COTS component is chosen, and needs to be repeated for each new version or release. Evaluation starts early and continues throughout the life cycle; feedback loops must allow for ongoing reevaluation of the COTS product and evaluations of new upgrades and revisions.

Developers need to ascertain two pieces of information about a component: whether the component itself is reliable and operates as advertised, and whether the system being developed will tolerate the component. The first assessment can be determined when the component is evaluated early in the life cycle. However, determining if the COTS product is suitable for a particular application, or if the system being developed will tolerate the COTS product, must wait until there is a system that will allow that type of testing. The COTS-based development life cycle must accommodate both types of evaluation as well as accommodate the acquisition and integration of COTS components.

It is also necessary to demonstrate that a COTS item has sufficient reliability in proportion to its importance to safety for its intended application. There are three sources for pertinent information: an examination of the vendor's development process and the associated product documentation, testing of the COTS software product, and an examination of the product's operational history. An adequate acceptance process should address all of these sources.

2.2.4.1 Examination of Vendor's Development Process and Documentation

Although a high-assurance software development process can significantly reduce the number of software faults, most COTS software components are sold with no warranty and are not typically subject to rigid development, verification, or testing processes. Often, the buyer has little opportunity to process information or documentation. Some system failures result from poor knowledge of the original requirements driving product development, and failure to test or make provision for the requirements in the upgrade. Consider, for example, the case of the Ariane-5 rocket. The time sequencing was based on an Ariane-4 rocket requirement that was not shared by the Ariane-5. This resulted in the switching off of two still healthy critical units of equipment, causing the rocket to self-destruct less than a minute into its first flight.

2.2.4.2 Testing

A test bed can be helpful in reducing risk, obtaining core competency, evaluating technology and forecasting, planning system evolution, determining requirements, and evaluating design and cost tradeoffs or alternatives. There is an important technical management consideration as well. The development facility, including hardware, development tools, and configuration management, must be ready before the first COTS product arrives for prototyping. Facility readiness fuels the

accelerated development COTS products provide, but moves the requirement for a fully implemented development facility to the beginning of the process. Determining COTS suitability requires a realistic target configuration with a strong system administration team in place from the start.

Based on the belief that independent certification is the only safe approach for component buyers, a safety program must incorporate black box component testing, system-level testing, and system-level fault injection [7].

Black Box Testing: Black box testing is conducted at the software component interface to demonstrate that input is properly accepted, output is correctly produced, and the integrity of external information (e.g., data files) is maintained. Such tests are used to demonstrate that the software functions properly. Black box testing is implemented to examine the internal logical structure of the software. However, this is the only kind of testing that can be used for characterizing the quality of a COTS component, since the evaluator generally does not have access to the source code for that component.

It is possible that independent black box testing can be used for COTS components. However, even for leading-edge COTS products, there is a lack of objective benchmark data, and often there is no data. It is a good strategy to develop a set of benchmark programs that exercise all critical attributes of the desired COTS product and run them on all products under consideration. Experimenting and testing are critical to feature discovery.

One approach to proper testing of a component is to choose testing activities that narrowly focus on particular qualities or attributes of the software. This demonstrates that the component is well qualified for its intended role. The goal is to test the quality of the component against the application system requirements (not necessarily against the vendor's claims).

Such an approach, however, ignores the fact that it is crucial to investigate the complete functionality of the COTS software. Too narrowly focused black box testing can fail to exercise significant portions of the code, which may be worrisome to developers attempting to certify components. Most COTS products contain more software than is needed to do the job for which they are evaluated. For a complex product, it may be difficult to demonstrate that these unwanted features do not interfere with system requirements.

An alternate approach is to conduct exploratory tests by invoking all the commands on menus and developing test harnesses to drive inputs throughout and beyond their expected range. One problem is that this may lead to a very extensive test program, as there is no easy way to bound what functions a COTS product might have. Functions being activated unexpectedly were responsible for the Ariane-5 rocket failure. Non-testing strategies for dealing with unwanted functionality will be discussed in Section 5.3.1.

A balance is needed between testing that is too narrow and testing that is too broad. Since there are more products and open questions than resources, evaluations should be focused and disciplined, and priorities defined and managed. It is advantageous to work on technology and

experiments that have direct relevance to an organization's business and projects to enable the moving of accumulated knowledge back into other projects for actual use.

A serious problem that even broad black box testing may not address is the potential for COTS software to contain unknown, incorrect functionality, including Trojan horses. A Trojan horse component appears to be benign and to behave as expected when invoked, but also will execute malicious functionalities. Only one test case may detect the Trojan horse, and this test case may not be covered during black box testing. Though incomplete, black box testing plays an important role in assessing component quality.

System Level Testing: Black box testing alone is insufficient to ensure that the component is a good match for the target. It is also necessary to determine how the target system will react to the component during system level testing, in which further testing of a COTS software item is performed within the context of the larger system. To ensure that a component is a good match for the target system, it is necessary to observe the component in the context of the full system to determine how well it will be tolerated. The goal of this type of testing is to determine acceptability or non-acceptability of the COTS component for use in a particular application.

To determine the acceptability of COTS components to the target system early in the process, an operational test bed using simulations for portions of the system not yet developed will provide this information.

It is crucial that the system safety requirements and the proposed role of the COTS product in a safety-related system be understood so that they can be incorporated into system-level tests. This is also the time to clearly categorize the COTS product system interactions and determine the rigor of subsequent qualification procedures.

System Level Fault Injection: It is necessary to determine whether the system can tolerate anomalies; i.e., how badly the system behaves if a component fails. This type of testing, known as fault injection, is a technique in which the tester injects a fault in the system to see how it will react. Testing the system by creating errors in it does not explicitly demonstrate the reliability of a component, but predicts how badly the system might behave if a component fails. When testing a COTS operating system, concerns include what the COTS operating system does when it gets corrupted and if the operating system itself may produce bad output. The fault injection technique for testing in such situations is to corrupt data moving to and from the operating system. The key to applying fault injection to data coming from the operating system is to determine how calls to the operating system return information. This allows the test to simulate a failure of the entity that produced the information. In this manner, the way in which the operating system reacts to bad input and output can be investigated. If a system can tolerate random failures, it will most likely tolerate real component failures. Since system level fault injection is an assessment of how well an application can recover after receiving bad COTS component output, the testing will make the system more robust, rather than merely determine why returned information is bad.

2.2.4.3 Operational History Associated with the Product

It is important to consider the stability of the COTS software and the degree of historical use. Real-time operating systems and kernels tend to be very stable because they have been used in hundreds of thousands of application environments for extensive periods of time. This combination yields confidence in a COTS component. In contrast, components that have a history of constant changes and untested paths are less likely to be stable.

Historical data can demonstrate reliability and increase confidence in the safety of the component. However, users do not often keep sufficient records to get enough historical data, and it is necessary to combine inspections, testing, and static analysis to make a full assessment.

2.2.4.4 Evaluation Closure

Completion of these evaluations, or testing activities, cannot provide absolute verification that the component works perfectly in all situations with all possible input. These evaluation steps – examination of the vendor's development process and documentation, testing, and operational history associated with the product – provide the system developer confidence in the product in direct proportion to the rigor of evaluation. It is more difficult to reach the decision to stop testing activities in COTS software than in custom software. However, the guidelines given herein will provide some direction to forming the test planning and test procedures in a way that will take into account the particularities of the system developed.

2.3 PROPOSED COTS DEVELOPMENT LIFE CYCLE

A proposed development life cycle for applications using COTS components is shown in Figure 1. The feedback loops indicate the need for iterative development during the system specification phase, and for COTS reevaluation during development and maintenance whenever revisions and updates of components need to be integrated into the system. In each case, the significance and scope of the revisions or updates will dictate the reevaluation necessary.

When COTS software components are used in a system, the software designer should be involved in the system architecture definition. The system architecture and requirements for COTS components must be considered in parallel. Though an initial system architecture may be proposed, requirements for COTS components and the accompanying market survey and product evaluation may require modification of the architecture. This iterative process continues until a system architecture has been designed that is appropriate for the available, qualified COTS components that will be used in the fielded system.

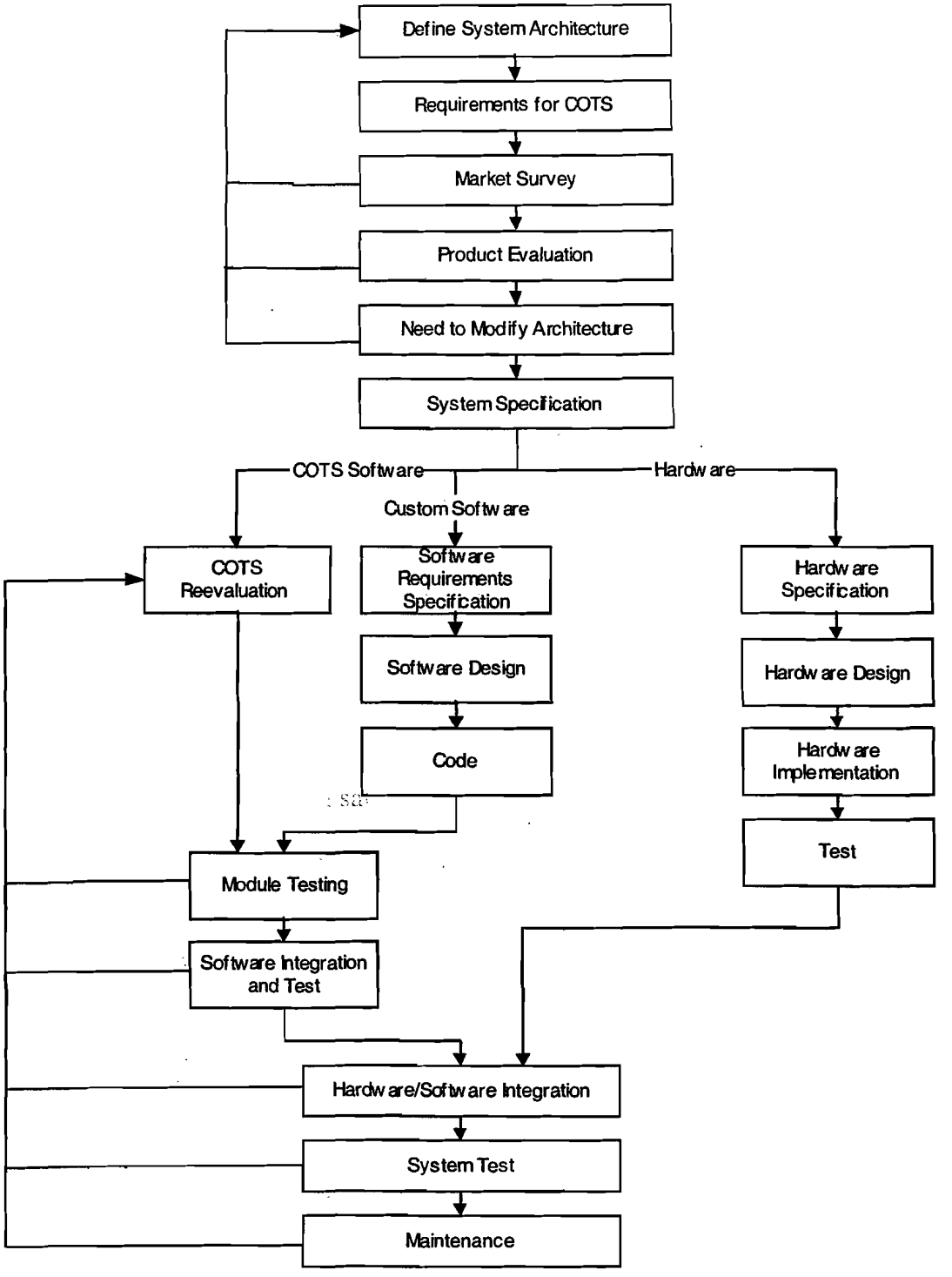


Figure 1. Development Life Cycle for Systems Using COTS Components

A system specification is then written and functions are allocated to hardware and software, consistent with the COTS decisions that were made in the systems analysis phase. Software development will need to accommodate both custom software and COTS components. In contrast to custom software, COTS components will not undergo the usual software requirements analysis, design, and coding phases. Instead, while the custom software is being developed, COTS components will be evaluated and tested. As described in Section 2.2.4, component evaluation will encompass the following:

1. Examination of the vendor's development process and documentation.

2. Black box testing (and some level of system testing and fault injection testing if an operational test bed is available to simulate the target system).

3. Examination of the operational history associated with the component.

Once module testing and evaluation is completed for both custom and COTS software components, software integration and testing will take place. When the hardware and software integration are finished, the system is ready for testing. The functional testing and fault testing that are part of the traditional system testing include the system level testing and system level fault injection portion of the component evaluation procedures directed specifically at determining how well the system tolerates its COTS components.

For as long as the system is in use, the maintenance phase will remain active. One aspect of system maintenance is accommodating the constant change, revisions, and updates expected when utilizing COTS components. This necessitates the use of feedback loops for reevaluating revised or updated components. All new components must be evaluated using the same component evaluation techniques described in Section 2.2.4. New components will need to progress through module testing, software integration testing, hardware and/or software integration, and system and fault injection testing before becoming a part of the fielded system.

3 TRADITIONAL SAFETY VERIFICATION AND VALIDATION METHODOLOGIES

This section examines the traditional method for performing a safety analysis known as system safety verification and validation (V&V). As described in the statement of work for this task, "The current process is essentially a subset of the system design and development cycle for safety-critical systems. The purpose of software safety V&V is to provide a means to incrementally increase the level of confidence in the safety-critical software being developed." This process ranges from safety V&V planning during early system development activities to safety verification and validation activities throughout software and hardware development and integration. Safety V&V planning documents should describe the actual safety V&V activities to be employed, the approaches and methods to be followed, testing strategies, deliverables, and accountability. Software safety verification takes place during requirements analysis, design and coding, module testing, integration testing, and system integration testing. Once software development is complete, a software safety validation is performed. At the end of system development, a system validation is required. These system safety V&V activities are illustrated in Figure 2 [8].

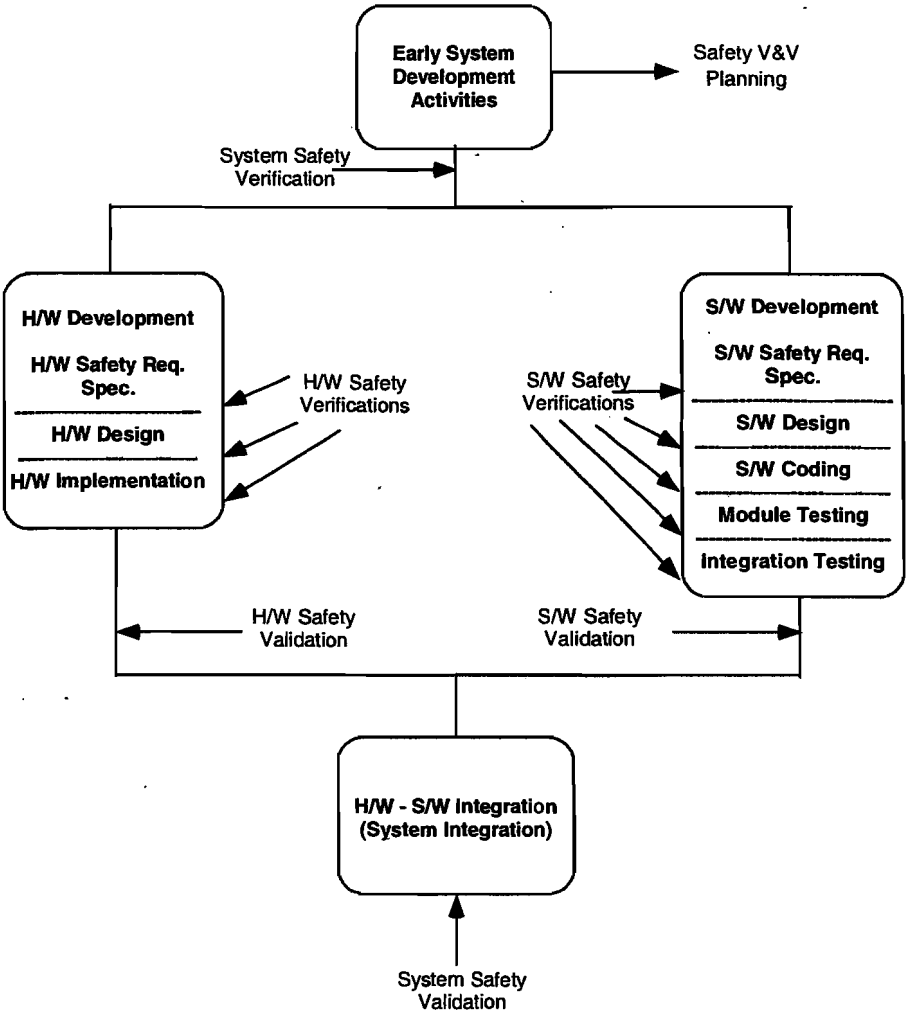


Figure 2. System Safety V&V Activities

3.1 CURRENT SAFETY VERIFICATION AND VALIDATION METHODOLOGY CHARACTERISTICS

According to Luedeke [8], the safety V&V methodology made available by the FRA to the railroad signaling industry is a recommended practice rather than a strict requirement. The methodology deals with system, hardware, and software safety-related requirements only. Hazard analyses are performed in the early stages of system development to identify potential hazards and associated risks – the intent being to establish safety requirements and alter the design accordingly during the early development stages. Then, safety verification and/or validation activities are performed to ensure that safety requirements are met and that no unsafe conditions are present in the system.

As there are no precise meaningful criteria or tests to determine that adequate levels of safety have been achieved, satisfactory safety requirements are based on the proper conduct of a well-structured development and safety V&V program. Evidence that all applicable safety requirements are met is measured in large part by determining how closely a developer has followed the recommended safety V&V practice.

The FRA does not presently certify or approve computer-based systems, as is done by other organizations like the Federal Aviation Administration. This approach does provide some latitude for the unique characteristics of COTS components. Nevertheless, the FRA needs to have a methodology in place to establish a consistent basis for demonstrating safety, improving and maintaining existing levels of safety for safety-critical systems, and establishing fault-tolerant architectures.

3.2 CURRENT METHODOLOGY PROCESS STEPS

Step 1: Preliminary Hazard Analysis (PHA). Hazard analysis and risk assessment are performed on the conceptual system in the early stages of system development. Fault trees are often used in this phase to graphically depict the functions that must fail for a hazard to occur – or the sequence of faults required to produce a hazard. The output of this step is a Preliminary Hazard List (PHL), which delineates hazards uncovered by the analysis. The PHA is performed to establish proper and complete safety requirements.

Step 2: Safety Requirements Specification. The Safety Requirements Specification must answer the question: What safety requirements are needed to deal with hazards? Since the main purpose of safety V&V is to demonstrate compliance of an entire system (hardware and software) with safety requirements, it is necessary to establish proper and complete safety requirements. At this stage, a peer review group will perform a system safety verification. The purpose of this verification step is to ensure that the specification accurately reflects the system safety requirements, as dictated by the hazard analysis, the user, and intended application of the system. Some iteration may be necessary until all the system safety requirements are properly covered.

Step 3: Safety Requirements Allocation. In this step, system safety requirements are allocated to hardware and software, and verification is performed to ensure that all system safety requirements are properly allocated.

Step 4: Software Requirements Specification. The Software Requirements Specification is examined to ensure that it includes the necessary specific safety requirements for software as dictated by the allocation of system safety requirements.

Step 5: Software Verification. Using peer reviews, analysis, inspection techniques, and testing; software verification is performed at each stage of software development. During software requirements specification development, verification is performed to ensure that the software requirements reflect the safety requirements as specified in the Safety Requirements Specification. During software design, the verification process ensures that the software design reflects the safety requirements defined in the software requirements specification. The hazards identified in the PHA can often be related to specific software components once the design is complete. The verification process continues during software coding, software module testing, and software integration testing to ensure that, in each phase, the safety requirements of the previous phase are met.

Software verifications are considered to be an incremental confidence-building activity in ensuring and demonstrating the safety of the software. However, they are also considered to be a necessary part of ensuring software safety. For instance, certain human errors that could result in unsafe software errors may not be found by a check on a final software product. One example is the preparation of a software requirements specification that contains incorrect safety requirements. Conducting a software verification directed specifically at the safety aspects of this specification can provide the best protection against such errors.

Step 6: Software Safety Validation. Once software development is complete, there is a review of previous verification activities to examine what was performed and what was found. Additional functional and black box testing is performed. Safety-related testing must demonstrate that safety-critical functions perform properly under normal operating conditions, systematic failure, random hardware failure, and external perturbations. Normal operation refers to normally anticipated operating conditions with no hardware failures and proper inputs over the expected range of environmental conditions. Systematic failure refers to human error occurring at various stages throughout system development. Random hardware failure includes single point as well as multiple hardware failures, including latent failures and common mode failures.[1] External perturbations include electrical and mechanical disturbances, abnormal environmental conditions, and unknown modifications.

Step 7: Hardware and Software Integration. Verification is used to demonstrate the safety of any hardware that relies, at least in part, on software to ensure safe operation. In addition, verification should demonstrate safe operation of the software under conditions of hardware failure or the circumstances outlined in Step 6.

---

[1] Common mode failures denote the failure of multiple components in the same way, such as a jammed valve. The term "latent errors" refers to present and potential errors not evident or active.

Step 8: System Safety Validation. This step demonstrates that the overall system complies with system safety requirements and is fit for use. Testing should address all relevant potential hazards defined in the System Hazard Analysis.

4 PROPOSED SAFETY PROGRAM PLAN

According to the Railroad Safety Program Plan (RSPP) found in the Federal Register [9], "System safety means the application of design, operating, technical, and management techniques and principles throughout the life-cycle of a system to reduce hazards and unsafe conditions to the lowest level possible, through the most effective use of available resources. The system safety approach requires an organization to identify and evaluate safety hazards that exist in any portion of the organization's 'system,' including those caused by interrelationships between various subsystems or components of that system. The organization then creates a plan designed to eliminate or mitigate those hazards. Where possible, the development of a system safety plan precedes the design, implementation, and operation of the system, so that potential risks are eliminated at the earliest possible opportunity. System safety plans are viewed as living documents, which should be updated as circumstances or safety priorities change or new information [is] available."

The relevant question is: In what way would a safety plan for COTS development differ from the safety plan generated for traditional development? A safety plan generated for traditional development would have to specify how the process steps outlined in the previous section would be implemented for a specific application. Since there will be necessary adjustments to the methodology outlined in Section 3 when COTS components are used in a system, process steps need to be defined for COTS-based software development. A safety plan generated for a specific application would then specify how these steps would be implemented for that specific application.

Though system development using COTS components precludes some of the verification steps outlined in the previous section (specifically, COTS software requirements specification verification, software design verification, and software coding verification), it is still possible to adhere to the intent of the RSPP when using COTS components.

The following sections note the alterations that occur in the current process made available by the FRA, concentrating on how the software module verification step must be augmented when using COTS.

4.1 PROCESS STEPS

Figure 3 shows the iterative life cycle for the component-based development approach presented in Figure 1 with safety V&V activities added.

Steps 1-3 of the proposed safety program plan will be similar to the recommended practices outlined in Section 3 as these steps are completed before any allocation to hardware and software has been decided, or before implementation options are considered. Theoretically, the decision of COTS or custom design has not yet been made. However, as a part of system-level requirements and analysis, complete system-level hazard analyses are particularly important as they provide the basis for determining the required system safety functions, some of which may be performed by COTS items. Hazard analysis is of great use in component evaluation, and in the selection or rejection of candidate COTS components. Even at this system-level phase, there

will be an iteration of the architecture, as described in Sections 2.2.1, 2.2.3, and 2.3, to accommodate the use of COTS components. Once the iteration is complete, a system requirements specification is written. All safety requirements will be included in this system requirements specification or in a separate safety requirements specification. Safety requirements allocation to hardware and software is in large part determined by the iteration that has taken place to determine where COTS products will be used.

Step 4: Software Requirements Specification. This document derives the software requirements from the system requirements for subsequent design and implementation. This activity does not have a parallel for the chosen COTS components. Software requirements, as well as design and implementation, will be hidden from the user of the COTS product. Therefore, evaluation activities need to provide the assurance that the portion of system requirements allocated to software allotted to COTS software components are met by the chosen COTS components.

Step 5: Software Verifications Using Peer Reviews, Analytical and Inspection Techniques, and Testing. These are recommended at each stage of software development [8]. All custom software in the system can still undergo verification at each stage of software development. Software verifications for COTS components can be performed only at the software module test and software integration test stages. A rigorous component verification process must be adopted for all COTS components to attain a portion of the confidence level provided by the missing verification activities during requirements specification, software design, and software coding.

Some component evaluation was performed early in the life cycle when the component was tested against requirements (including safety requirements); therefore, some verification has been accomplished previously. This evaluation testing should be carefully assessed and repeated where necessary. Software system integration testing is crucial because the component also has to be judged on how well the overall system will tolerate it.

Steps 6-8: Software Safety Validation, Hardware and Software Integration, and System Safety Validation. These are performed for custom and COTS software and hardware, as they would be with custom development alone. The software safety validation and system safety validation steps will require extra emphasis on testing the effect of the COTS components on the full system.

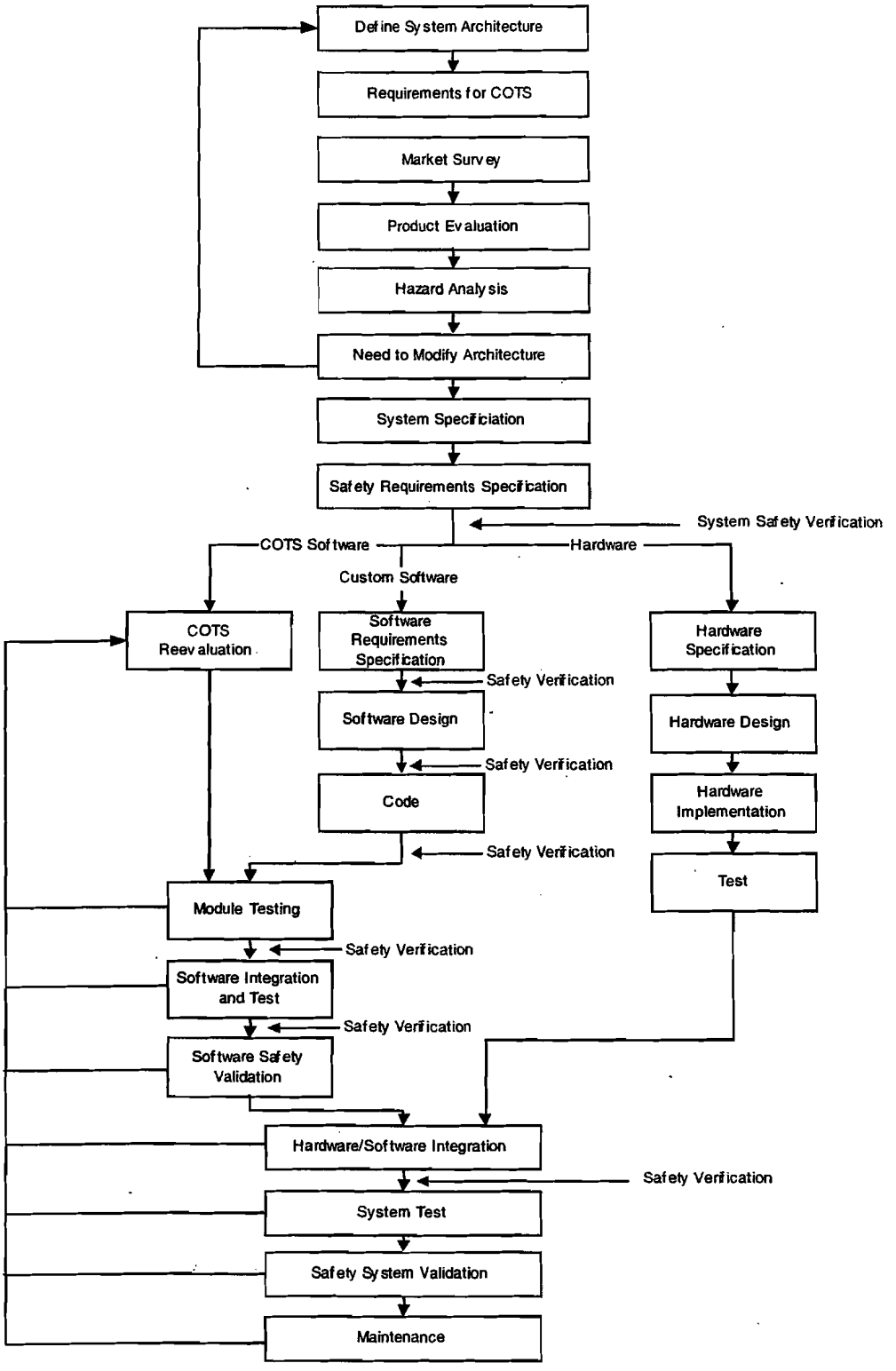


Figure 3. System Safety V&V for COTS-Based Systems

5. APPLYING COTS AT THE MICROPROCESSOR/OPERATING SYSTEM LEVEL

5.1 IMPLICATIONS OF USING COTS OPERATING SYSTEMS

5.1.1 ALTERNATIVES

An operating system is a layer of software that manages a computer's resources and presents the user with an interface that is easier to program than the underlying hardware. However, rather than use a commercial operating system, the software developer could design a real-time control system with no operating system layers between the real-time application and the hardware that it controls. The first question is whether to use a commercial operating system at all. Early real-time systems were written with no operating system layers between the real-time application and the hardware that it controlled.

There are a number of reasons why developing real-time control systems this way is discouraged. Today's real-time control systems often have to perform many tasks and are usually built on top of real-time operating systems to facilitate the handling of these tasks. Otherwise, significant time is spent developing services; such as schedulers, timers, device drivers, and task synchronization mechanisms generally provided by an operating system. Early real-time systems were most often written in assembly language; which is difficult to understand, maintain, or reuse on new projects since it is specific to a particular type of machine architecture. In addition, device drivers must be rewritten whenever hardware is updated. Current real-time applications often require complex services like networking, graphical display and windowing systems, as well as file and database systems. These are often difficult and time-consuming to write.

Because of these factors, using a commercial operating system may be one instance where a COTS product provides more safety than a custom built system. To decrease time to market, many companies are moving from the traditional method of developing embedded systems. Instead, they are using modular operating system components and focusing on developing their custom applications instead of the underlying operating system services.

5.1.2 BENEFITS

With the increasing complexity of hardware devices, operating systems that provide device drivers are becoming a necessity. In the past, hardware manufacturers would provide documentation of low-level interfaces to their devices so that software developers and real-time engineers could then write custom device drivers. This is no longer the case. Now, hardware manufacturers often provide device drivers that will interface with general purpose operating systems (GPOSs) and are increasingly reluctant to provide documentation of the low-level interfaces to individual software developers. Many hardware manufacturers offer device drivers only for GPOSs. Device drivers for standard devices, such as serial ports and hard drives, are often included as part of GPOSs. These trends are increasingly forcing engineers to choose GPOSs. As processor speeds have increased, GPOSs like Windows NT have experienced increased usage in real-time systems as they have become able to meet real-time requirements. Most GPOSs are designed so that the scheduling algorithms ensure fair allocation of the

processor, minimize response and turnaround time, maximize throughput, and process efficiency. More real-time software developers are using GPOSs because they are often less expensive and better supported than their real-time counterparts [10].

Real-time operating systems like Windows CE also try to satisfy precedence, resource, and time constraints for applications that need this functionality. Windows CE is targeted to a different market and makes different feature-versus-size tradeoffs than Windows NT. Generally, Windows CE is targeted at smaller embedded applications where low power, size, mobility, or the cost-of-memory are overriding issues. Hand-held devices and other high-volume products with low unit costs are classic examples of the target marketplace. These are developed on the premise that increased engineering development cost will yield lower unit production cost.

Many real-time users continue to choose Windows NT. It has a strong presence in the marketplace; at least three times greater than other alternatives. Software support, development tools, documentation, training courses, and other educational materials are plentiful. Windows NT supports a wider and richer set of third-party software than Windows CE.

There also exist a large number of Windows programmers, thus making it easier to find trained people to maintain the system over the course of its life cycle. Finally, PCs are generally less expensive than specialized hardware. The majority of PC vendors ensure hardware compatibility with the Windows platform. The development platform and target system often use the same operating system. This allows most of the software development to be completed before the target hardware is available.

Building on customer preference for Windows NT functionality but recognizing that embedded and real-time systems may not be well served by the full functionality of Windows NT, Microsoft recently developed the Windows NT Embedded operating system. Windows NT Embedded is generally targeted at the more complex embedded applications. This product is a version of the Windows NT operating system that can be customized by the user so that only the necessary components will be included in the target operating system. Using authoring tools, the user begins with a sample operating system configuration and then customizes the device by adding and removing functionalities and importing additional custom components. Benefits of this approach include the incorporation of the Win32 application programming interface, increased interoperability with other applications, drivers and services, error recovery, and Windows NT security features. The embedded system developer does not need to develop and maintain a proprietary operating system. This should lead to lower engineering development costs. Finally, since functionality can be restricted, a smaller and more focused operating system can be obtained.

5.2 COMMERCIAL OPERATING SYSTEM ISSUES

There are several issues associated with commercial operating systems use that must be addressed prior to system integration. Commercial GPOSs may be insufficient for applications requiring real-time features. The lack of access to operating system source code is a concern, particularly for safety-critical applications.

5.2.1 Real-Time Operating System versus Non-Real-Time Operating System

Railroad train control systems have real-time requirements because they must satisfy explicit (bounded) response-time constraints or risk severe consequences, including failure. Response time is the time between the presentation of a set of inputs to a software system and the appearance of all the associated outputs. A typical real-time control system collects and processes the data and sends control commands to physical devices such as actuators.

Desirable attributes for real-time operating systems are predictable response times, priority-based scheduling, and preemptive control. Predictable response times under all load conditions ensure that applications respond to external events in a predictable fashion, regardless of what other tasks the system is handling. Priority-based scheduling of multiple tasks ensures that higher priority tasks will always execute before lower priority tasks. An effective real-time operating system requires consistent and prompt priority-based scheduling of time-critical tasks and low system overhead. Preemptive control ensures that deadlines are met by allowing lower priority tasks to be preempted by higher priority tasks. Characteristics commonly found across all commercial real-time operating systems include guaranteed response to external interrupts; mechanisms to support input/output (I/O); interprocess communication mechanisms; low system overhead; access to very precise timing reference; user control of system resources; and development support.

The function of the underlying operating system is to satisfy the requirements of a real-time control system (with its real-time-related safety requirements). One important goal of a railroad supplier's component evaluation process will be to determine whether a train control application requires a real-time operating system. This will necessitate a careful examination of system requirements, including timing requirements.

5.2.2 Access to Operating System Source Code

A persistent obstacle to the acceptance of commercial operating systems, particularly for safety-critical applications, is the lack of access to the source code. If the train control supplier had access to the operating system source code, the traditional V&V process recommended by the FRA could be followed. This would permit white box and black box testing of the source code. Any potential issues with the operating system could be identified, and any incorrect code could be modified or eliminated. This approach will achieve results. While Windows products do not allow access to source code, Linux and other Unix variants do, and have a wide circle of supporters and users, despite the expenses involved. Purchasing systems such as QNX and VxWorks is more expensive than purchasing their Windows counterparts even without an operating system for each target. Though some source code is freely accessible, the majority needs to be purchased separately at a considerable cost. Even then, the operating system code is difficult to understand. Only in theory could a case be made to verify it.

Considering the cautions about modified COTS detailed earlier in this report, the danger in adopting the MOTS approach is that the developer could realize the worst aspects of the custom code and COTS worlds. Still, there is an ongoing debate between proponents of both approaches. Supporters of the Unix family of operating systems cite the sheer size of the

Windows NT source code and its growth with each succeeding revision as evidence of system/code unreliability. "People who build reliable systems don't radically change the system very often," says Brian Croll, of Sun Microsystems, Inc. They argue that critical software will be more reliably interfaced with a system whose source code is available (e.g., Linux) or a system that is simpler, more reliable, and better documented [11].

5.2.3 WINDOWS NT ISSUES

Microsoft designed Windows NT as a GPOS and has stated that it is not a real-time operating system. Windows NT cannot be used if sub-millisecond response time is required; but if the required response time for a system is greater than 5 milliseconds, then Windows NT may be a viable candidate as an operating system. A number of application developers are addressing this shortcoming by developing products that work in tandem with Windows NT Embedded, and provide better real-time response. For example, the VenturCom Real-Time Extensions (RTX) for Windows NT Embedded shorten the response time resolution from greater than 5 milliseconds, to 30-50 microseconds.

Window NT's interrupt handling architecture is another potential problem. The interrupt is a mechanism that informs the system when an important event occurs. Input/output, hardware devices, software timers, and exception handlers, may generate interrupts. Windows NT's interrupt handling architecture traditionally has been very slow at handling interrupts, although performance continues to improve. The Windows NT Embedded product will have different interrupt handling characteristics. The VenturCom RTX also promises enhanced interrupt handling. Whether the interrupt handling capabilities will be factors for operating system selection is a function of the control system requirements. Potential problem areas must be identified and understood. The component evaluation phase of the development life cycle must compare the capabilities provided by the operating systems, and extensions being evaluated, with capabilities required by the real-time systems.

System overhead is another issue that must be addressed when using Windows NT. Because Windows NT is compatible with a range of applications, it consumes large amounts of system overhead. This is the primary reason that Windows NT takes longer than other operating systems to perform functions like context switching, task switching, and interrupt handling.[2] System overhead should be greatly improved for Windows Embedded NT applications, like VenturCom RTX.

Stability is a persistent concern with Windows as well. Real-time control systems are often required to operate for long periods of time without interruption or human intervention. An operating system crash may cause an entire control system to fail. When used as a general purpose desktop operating system, Windows NT sometimes crashes unexpectedly. This causes many engineers to distrust the stability of Windows NT and question its acceptability as a real-time operating system. Still, Windows NT's stability is difficult to determine. Since the operating system supports a large number and variety of third-party software and hardware, the

[2] Context switching is the process of saving and restoring sufficient information for a real-time task so that it can be resumed after being interrupted.

actions and interactions of third-party devices and applications may actually cause problems that are blamed on Windows NT. Once again, using Windows Embedded NT will undoubtedly improve the stability of the operating system as inessential features and capabilities of Windows NT are eliminated from the target operating system.

If systems are networked, security issues may arise. The evaluation process should look carefully at Windows NT security features, identify problematic areas with respect to security, and guard against them when implementing and deploying systems.

Another concern of many COTS users is the handling of updates and revisions when multiple COTS products are used. However, updates issued by manufacturers of COTS products are usually backwards compatible with components with which they have been designed to interface. This is a necessary business strategy. If the new update were incompatible with any of the other components in a system, the customer base would not use the upgrade.

Looking at Windows NT's Intel Pentium microprocessor, revision incompatibility does not appear to be a major issue. Windows NT updates occur once or twice a year, consisting mainly of corrections for software defects, and should increase the reliability of the operating system.[3] It is usually advisable to wait for the first service pack (corrected revision for software defects) to be issued before incorporating a new revision. Windows NT upgrades are always 100 percent backward compatible. Processors are upgraded about every 2 to 3 years and are compatible with the latest version of the Windows NT operating system and also backward compatible to several previous versions. In general, COTS products are designed to support a number of previous versions but will eventually stop supporting old versions of components. The system using COTS components should be upgraded periodically to maximize benefits. The practice of COTS suppliers supporting previous versions will result in timely, orderly upgrades.

5.3 RECOMMENDATIONS

If the decision is made to incorporate a COTS operating system into safety-critical applications (such as train control), there are other safety concerns to address. One concern is the impact of unwanted functionality in a COTS component. Another is the fault tolerance of the COTS architecture in the event of a COTS failure. These two concerns are discussed in the following sections.

5.3.1 PROTECTION AGAINST UNWANTED FUNCTIONALITY

If the decision is made to incorporate COTS components, appropriate methods to control the COTS component should be employed in order to address persistent safety concerns. One method for achieving this is to use only components that satisfy the user's expectations without providing unwanted functionality. In order to support this goal, complex COTS software products can be customized to provide only the desired functionality. For example, the user can tailor Windows NT Embedded so that only the required components are chosen for inclusion in the target operating system, as described in Section 5.1.2. Hazard analysis techniques, such as

[3] Major upgrades that introduce new features and represent major changes in the operating system result in a rename for the product (e.g., Windows NT, Windows 2000).

Functional Hazard Analysis and Hazard and Operability Studies, can be used to assist in determining which functions to avoid because of corrupting essential data or danger in the system context. Safety analysis techniques, like fault trees, also can be used to investigate how the software contributes to system safety. These techniques may be used to identify the features of the COTS software that may need to be disabled if they prevent critical functions from being executed [5].

Another technique that employs the lessons learned in the component evaluation process is to install a software "wrapper" around a component. A wrapper is an instrumentation layer placed around an operating system to control all interactions between an application and the operating system. This prevents undesirable input and output from interfering with system functionality. Wrappers are not used to directly modify a component's source code, but are designed to indirectly modify and limit a component's functionality. One good use of a wrapper is to handle exceptions generated by the operating system. Notifications of error occurrences are exceptions, and they are often generated when errors or other anomalous conditions arise during program execution. When applications fail to handle such exceptions, they crash. A famous example of this phenomenon occurred in September 1998 aboard the U.S. Navy aircraft carrier Yorktown. A petty officer inadvertently entered a zero into a shipboard database. The resulting database numeric overload caused the Windows NT operating system to crash, leaving the ship dysfunctional and having to be towed into port. This is precisely the kind of exception that a wrapper would prevent.

There are two types of wrappers. One prevents certain input from reaching a component, preventing the component from executing on the input, and limiting the output range of the component. The other captures the output before it is released by a component and passes the output on only if it satisfies certain constraints. Both types of wrapper can be implemented for the same component. However, wrappers cannot protect against events unanticipated by developers.

5.3.2 Development of a Software Fault-Tolerant Architecture

Safety-critical systems in the railroad industry traditionally have been designed "fail-safe" by employing "vital relays" and other discrete components with well-defined failure modes. Even in these vital relay-based systems, safety is not absolute. Rather, it is probabilistic in nature since there is still a finite, but extremely low probability that an unsafe failure could occur.

For COTS components to be used safely and effectively, a software fault-tolerant architecture should be implemented. When implemented correctly, this type of architecture incorporates increased safety. A fault-tolerant architecture also may allow developers to modify existing applications and incorporate new or upgraded COTS software components easily, affordably, and reliably.

For hardware systems, fault-tolerant architectures often employ hardware redundancy and majority voting techniques. These are ineffective against software faults. Another well-established engineering approach to guard against the failures often found in complex systems is

to provide a simpler back-up system with assured reliability. In this simpler, high-assurance system, COTS components with uncertain reliability would not be used.

The requirements for these two techniques are usually different because the former is often associated with high performance systems while the latter is employed in the design of high reliability systems. The high reliability approach works well for systems with states (such as feedback control and command-and-control applications) that can be monitored.

Overall, this approach may necessitate a more complicated design for the system. Essential portions of the current train control systems could be retained as a backup until the FRA and the railroad supplier industry gain more confidence in the automated processor-based systems. This would not eliminate the need for careful evaluation, thorough testing, and V&V of new systems, but might provide help until the FRA and the railroad industry obtain confidence in the automated systems under actual operational conditions [12].

6 FUTURE WORK

An initial concentration on microprocessor and operating system COTS components for train control system development platforms has merit. Most COTS implementation takes place at the software component level, including graphical user interfaces, operating systems, and databases. These large components are being used quite widely in embedded real-time systems.

Further development or system upgrades may be used once initial system experience has been evaluated. At that point, other issues regarding integration, bridge software, and component-based architectures should be examined in detail.

6.1 INTEGRATION ISSUES AND BRIDGE SOFTWARE

Despite the goal of plug-and-play components, multiple COTS software components seldom integrate easily into an existing system. Often, COTS products have been designed to provide "stand-alone" capability and have not been designed with integration as a requirement. Because of this, extensive software development is often needed to allow COTS components to interface properly. Bridge software or glue code is often developed to integrate COTS products and fill in missing functionality.

6.2 COMPONENT-BASED ARCHITECTURES

A well-defined, standard component-based architecture provides a number of valuable services to the developer of a COTS-based system. Ideally, it will specify the following: (1) how each component must make its services available to another; (2) how to connect one component to another; (3) which common utility services can be assumed to be provided by the infrastructure supporting the component model; and (4) how new components announce their availability to others. A standard component-based architecture can provide developers with a programming approach that will facilitate the effective use of COTS components. Investigation of such architectures and the more complex integration issues that the use of multiple COTS components introduces are worthy topics for further study.

There are two major component-based architectures: Common Object Request Broker Architecture (CORBA) and Distributed Common Object Model (DCOM). CORBA is a specification of a standard architecture, not an implementation. However, while CORBA defines a standard, there is great latitude in many of the implementation details. For example, two Object Request Brokers (ORBs) may be CORBA compliant but they may possess significantly different features and capabilities. Users must learn a specification, become familiar with the way vendors implement the specification, and become familiar with their value-added features (often necessary to make a CORBA product usable). There is a large and growing number of CORBA implementations available in the marketplace from most major computer manufacturers, as well as independent software vendors.

The CORBA industry standard has been flexible in response to changes in market conditions and technology advances. In comparison, development of formal standards often progresses slowly. However, one problem with CORBA's flexibility is that changes to the specification, while

technically justified, have resulted in unstable ORB implementations. While not a perfect solution to component interoperability, CORBA is an important step in achieving this goal.

COM is a software architecture that defines a standard for component interoperability. Therefore, components developed by different software vendors may be combined into a variety of applications. COM is a general architecture for component software, but it is not a specification for how components are structured. Instead, it is a specification for how components and applications interoperate. In COM, components interact with each other and with the system through collections of functions, called interfaces. An interface is a contractual way for a component to expose its services to other components or to the system. These interfaces are unchangeable and function as contracts between components. This facilitates updates to new versions as long as the interface between the two components does not change. The new version of the component is transparent to the system. If a component upgrade increases functionality or otherwise requires a new interface with the world, then a new interface must be defined. This new interface is assigned a unique identifier, and the way that clients access the component will have to be changed. This approach results in a stability that more flexible architecture implementations lack. There are ORB products that are not compliant with either CORBA or COM, but they tend to be vendor-unique solutions affecting system interoperability, portability, and maintainability.

REFERENCES

1. Goode, Jane, and Heckathorne, Jasjit. *Review of Commercial Off-The-Shelf (COTS) Software System Issues*, Draper Laboratory, Cambridge, Massachusetts, January 2000.

2. Carney, David J. and Oberndorf, Patricia A. *The Commandments of COTS: Still in Search of the Promised Land,* Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1997.

3. Foreman, John. *On the Front Lines of COTS – Lessons Learned, Speculation for the Future*, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1998.

4. McDermid, John. "The Cost of COTS," *Computer Magazine*, Vol. 31, No. 6, June 1998.

5. Fox, Greg, Lantner, Karen, and Marcom, Steven. "A Software Development Process for COTS-Based Information System Infrastructure: Part 1" *CrossTalk*, May 1998.

6. Parra, Amy, and Kraft, Steve. *SEL COTS Study*, Software Engineering Laboratory Series, NASA Goddard Space Flight Center, Greenbelt, Maryland, August 1998.

7. Voas, Jeffrey. "Certifying Off-the-Shelf Software Components," IEEE Computer Magazine, June 1998.

8. Luedeke, J.F. Analytical Methodology for Safety Validation of Computer Controlled Subsystems, Volume II: Development of a Safety Validation Methodology, DOT/FRA/ORD-95/10.2, 1995.

9. United States Federal Register, Standards for Development and Use of Processor-Based Signal and Train Control Systems; Proposed Rule, Vol. 66, No. 155, August 10, 2001.

10. Kyemba, Heather. *Analysis of the Applicability of a General Purpose Operating System for Time-Critical Tasks*, Draper Laboratory, Cambridge, Massachusetts, May 1999.

11. Martellaro, John. "Going to War with Windows NT: Another Vietnam?" Mac OPINION, September 14, 1998.

12. Sha, Lui, Goodenough, John B., and Pollak, Bill. "Simplex Architecture: Meeting the Challenges of Using COTS in High-Reliability Systems," *CrossTalk*, April 1998.