



---

National Research  
Council Canada

Conseil national  
de recherches Canada

Institute for  
Information Technology

Institut de technologie  
de l'information

---

**NRC - CNRC**

---

***COTS Software Integration:  
State of the art***

Mark R. Vigder  
W. Morven Gentleman  
John Dean

Software Engineering Group  
January 1996

Copyright © 1996 by  
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright © 1996 par  
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

## TABLE OF CONTENTS

<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>2. CHARACTERISTICS OF OFF-THE-SHELF BASED SYSTEM DEVELOPMENT.</b>	<b>1</b>
<b>3. OFF-THE-SHELF COMPONENTS: STATE OF THE ART.....</b>	<b>2</b>
3.1 CURRENT PRACTICES.....	3
3.1.1 <i>Buy-and-adapt System</i> .....	3
3.1.2 <i>Integrating components</i> .....	6
<b>4. EXPERIMENTS WITH OPEN SCRIPTING ARCHITECTURE (OSA).....</b>	<b>8</b>
<b>5. RELATED WORK.....</b>	<b>10</b>
<b>6. ISSUES WITH COTS SOFTWARE INTEGRATION.....</b>	<b>12</b>
6.1 PROCUREMENT/DEVELOPMENT PROCESS.....	12
6.2 UNDERSTANDING AND EVALUATING COMPONENTS.....	14
6.3 EVOLUTION OF SOFTWARE.....	14
6.4 ARCHITECTURAL ISSUES.....	15
6.5 ROLE OF STANDARDS.....	16
6.6 EMBEDDED SYSTEMS.....	16
<b>7. FUTURE RESEARCH.....</b>	<b>17</b>

## **1. Introduction**

The Software Engineering Group of the National Research Council (NRC) is currently undertaking a research project into the implications of using off-the-shelf (OTS) software to build long-lived military systems. The purpose of this project is to:

- Determine qualitatively the advantages and disadvantages of using OTS software in the construction of systems.
- Identify the types of systems that can benefit from the use of OTS components and the types of OTS components that can be used within these systems.
- Develop criteria for evaluating OTS components.
- Identify problems with the existing software and system development and procurement process that inhibit the effective use of OTS software.
- Investigate technologies and architectures that enable the use of OTS software.

This paper presents some initial results and observations regarding the OTS software usage. These results are based on the following activities:

- A series of interviews was conducted with personnel within DND who are involved in procuring or maintaining systems that contain off-the-shelf components.
- Round table discussions and interviews with researchers and commercial software developers who are interested in component based software.
- Review of the literature to determine the current state of the practice and current state of the research in building systems from OTS components, building software components, and open standards that enable the use of off-the-shelf software.
- Experimentation with different technologies and standards that are designed to enable the use of off-the-shelf components.

## **2. Characteristics of off-the-shelf based system development**

Building systems from off-the-shelf software components is an instance of a type of software re-use. It differs from other forms of re-use in that the system developer buys the components (usually without the source) from third party

developers and then integrates the components into the system. Characteristics of this approach to system development include:

- A component software product is designed to be sold in many copies to multiple customers with minimal changes.
- Pre-existence is one virtue - not only because it can shorten delivery schedules but because it means the customer can use pilot studies to rethink "requirements" and to investigate deployment problems.
- Honing in the marketplace, especially a competitive marketplace, improves specification, design, and implementation in ways that a waterfall development cannot anticipate.
- Vendor is responsible for ongoing support and maintenance - but this implies customer must accept upgrades
- No single customer has control over specification, schedule, or evolution.
- The specialized nature of the OTS product allows the customer indirect use of the rare skills of designers and implementers.
- Access to source code is unusual.
- The development process used may not be your favorite nor appropriate for easy configuration management and control.
- Internal documentation may be non-existent or not accessible.
- Technical information, especially as to limitations, performance, or resource consumption, may never have been collected.
- User level documentation, customer documentation, and training may be well developed.

Depending on the source of the components, they are referred to as Commercial off-the-shelf (COTS), Military off-the-shelf (MOTS), Government off-the-shelf (GOTS), etc.

### **3. Off-the-shelf components: state of the art**

As part of this research we have talked to numerous military, government, academic, and industrial people involved in the development of COTS components, or building systems from COTS components, and surveyed the literature to identify problems other organizations had had when building systems from COTS components [BRO95,NAS95]. The purpose of this survey was to understand the current practices of DND in terms of COTS software, determine the experiences and attitude of those responsible for software systems, and to learn what kinds of problems (and solutions) people had experienced in relation to COTS software integration.

Within DND we surveyed eleven different groups or organizations and eighteen different individuals involved in about fifteen projects. The projects were primarily Command and Control systems or information systems.

There was no attempt to gather quantitative data. Very few useful metrics were available and it is unlikely that these would have any relevance across different projects. Thus the data gathered is qualitative and anecdotal.

### **3.1 Current practices**

Within DND all the systems looked at as part of this study were either information systems or command and control systems. We did not look at weapons systems but are aware of a number of systems that use OTS software, particularly where it is embedded within specialized OTS hardware.

There are different ways of using off-the-shelf software. The primary approaches we found within DND are the following:

- Buy and adapt. The buy-and-adapt model is characterized by acquiring a single complete working system that satisfies most of the requirements of the acquisition agency and adapting and extending it for local needs. The adaptation of the system is done by extending it through add-ons, interfacing with other applications, or modifying the off-the-shelf application through source code changes (But then is it really "off-the-shelf"?)
- Component integration. The component integration model of software development builds software systems by integrating a number of off-the-shelf components where each component satisfies some of the requirements of the system. This model usually depends on the use of some "gluing technology", which may be unrelated to the components, to provide an interface between components.

#### **3.1.1 Buy-and-adapt System**

The buy-and-adapt model of OTS reuse builds systems by purchasing an application which satisfies most of the system requirements and then extending and tailoring the application to satisfy local requirements. This model of development was used for both command and control and information systems. The applications bought were both military off-the-shelf (MOTS) and commercial off-the-shelf (COTS).

Within DND we observed two methods for modifying and extending systems:

- *API's*. Most of the systems which were bought and adapted had some kind of an API. The developer can write a controlling program that calls the COTS component API as required. Typically this involves writing a "wrapper" around the OTS component to isolate the workarounds and extensions and provide a somewhat higher level of abstraction to the component's interface. The developer then writes the main program that controls the sequence of execution including calls to the OTS component (Figure 1a).

- *Modifying source.* If an OTS system does not satisfy all requirements then the supplier (or a contractor with access to the source code) can modify the application to satisfy the local requirements. However, once modifications to the source code is done the acquisition agency no longer has an off-the-shelf component but rather has a one-of-a-kind system. Using this approach there is a risk that these changes will become orphaned and the vendor will not support them during the normal course of upgrading the product.

The problems we have seen with systems that have been bought and adapted are:

- Limited source of supply. One argument for using COTS is that competition between vendors will drive prices down and improve quality of the systems. For many MOTS software applications there is limited choice of systems for purchase with minimal ongoing competition and these arguments do not apply.
- New releases of OTS component. Replacing older versions of COTS software with newer releases is difficult. New requirements or new hardware may preclude continued usage of the older version. Extensions and modifications made to the previous version must be re-integrated into the newer system. We saw three approaches to this problem:
  - Assume (hope?) the API and data formats of the new releases will not change significantly.
  - If extensions are added to the OTS software, have the vendor certify the changes as being compatible with all future releases.
  - If modifications have been made to the original source code, contract with the original developer (or someone with access rights to source) to make modifications to new releases. There is potentially a high risk (and high expense) associated with this approach: a complete process of analysis, development, documentation, and testing must be performed for the one instance of the product.

Within the commercial world the use of API's and source code modification are used but there is also a large amount of interest in using a concept known as *frameworks* . A framework is a large scale component or an application which is designed to be extensible and integrated with other frameworks. With a framework one is not buying a component called through an API, but rather an implemented architecture inside of which one can embed extensions using techniques such as plug-ins and inheritance. The difference between traditional API based components and frameworks is shown graphically in Figure 1. In the traditional model (Figure 1a) the custom code defines the architecture and the COTS component is embedded in and called by the custom code; in frameworks (Figure 1b), localizations and extensions are embedded inside the framework, use the frameworks architecture, and are called by the framework.

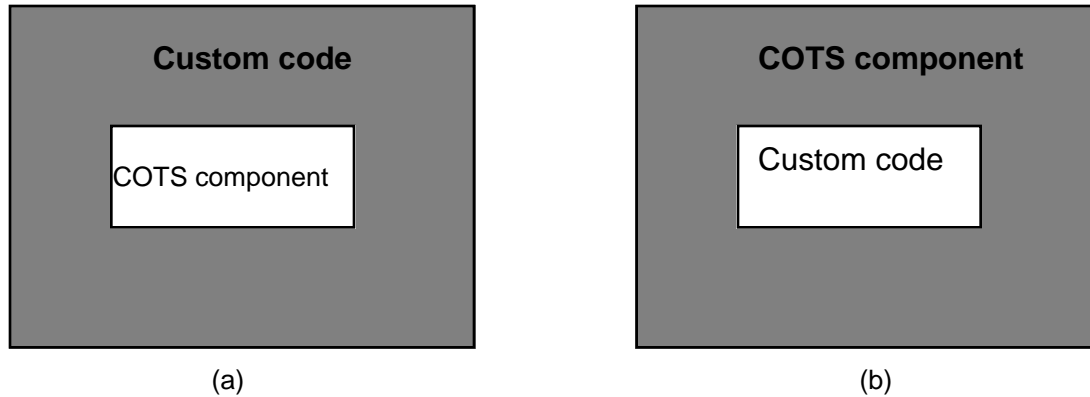


Figure 1.

- (a) Traditional model: COTS embedded inside custom code.  
 (b) Custom code embedded inside COTS.

Frameworks can be constructed to provide generic services (such as the OpenDoc framework which provides an open architecture) or they can provide services within a specialized application domain, such as to the financial services industry, manufacturing, health services, etc.

A framework can be tailored and extended in a number of ways:

- **Plug-ins.** Developers can add functionality to an application by writing a "plug-in". A plug-in notifies the framework of its capabilities and services and the framework calls the plug-in as required. Effectively this reverses the traditional role of a component and the custom code: rather than the custom code calling the component through an API, the framework calls the custom code that is implemented as a plug-in.
- **Scripting.** A script is an executable fragment of code which is dynamically linked to components of the system. A script can be used to extend the behaviour of a component (by having the component execute the script), or it can be used as a coordination mechanism to integrate two or more components (by providing the "glue" for linking the components together). Over the last few years, numerous languages designed specifically as scripting languages have been developed and are being used on a commercial basis (e.g., ObjectREXX, Visual Basic, AppleScript, JavaScript, tcl, Perl, Python, etc.) Currently there are two rival scripting architectures which are competing for market dominance: the Open Scripting Architecture (OSA) which is supported by IBM, Apple, and most of the computer manufacturers; and OLE Automation which is supported by Microsoft.
- **Inheritance.** Inheritance allows specific parts within a component to be specialized and modified. Current object-oriented standards allow inheritance to be used by developers who only have access to executable binaries and not to the source code. In contrast to scripting, which is often done by end-users or their support staff, inheritance requires a deep knowledge of the architecture of the framework and would normally be

applied only be professional developers. Examples of this approach can be found in the CORBA object model and DSOM from IBM.

Within the commercial world building systems from frameworks can be viewed as a "just-in-time" programming model where components move along an assembly line from the developer through the integrator to the end user, and functionality is added at the latest possible point along this line. Component developers build large-scale components that provide services that appeal to a large market; integrators extend and combine the components to build systems; and end users and their support staff tailor the system for local needs.

Although frameworks are being developed (or at least talked about) within many commercial application domains there is currently limited activity in the military domain. There is an initiative at DoD's ESC Software Center (described in [BRO95]) which is being developed along the line of frameworks. The objective is to define a set of product lines each with its own evolving architecture and continuing qualifying COTS products. A client approaching ESC for a solution will be sold one of these standard architectures tailored to the clients needs. By this means they intend to have the capability to deliver systems to their clients quickly and with very little development effort. The price they pay for the short delivery time and low cost is that they do not expect to be able to satisfy 100% of their clients requirements, but they do expect to satisfy enough of the requirements that they are considered a cost effective solution.

Another example of a framework based approach is the DIGEST GIS system being developed at DREV. The GIS system is the framework to which plug-ins can be added to extend functionality. The objective is for development of both the GIS framework and the plug-ins to be taken over by commercial developers.

### **3.1.2 Integrating components**

The integration approach to COTS usage involves the developer buying two or more separate COTS software packages and integrating them into a larger system.

A component to be integrated can be packaged in many different ways. Within the systems we studied we observed the following component types:

- *Procedural libraries.* Component is delivered as a set of library routines which are linked in at build time.
- *Legacy applications.* Application is part of an organizations structure and workflow and must be included as a component within the new system.
- *Off-the-shelf applications.* Component is delivered as a stand-alone application (which may or may not have open interfaces and data formats.) Integration can take different forms, such as API calls, shared data in standard formats, "screen-scraping", event passing, drag-and-drop, etc. The most common form of integration observed was the use of shared databases and shared files.

1/25/96

- *Tools.* Tools are a mechanism to automatically build source code. An example is a GUI builder. Tools typically work by having the developer describe the system using the tool's language. The tool then generates source code that can be compiled and linked with the other components of the system.
- *System services.* Operating Systems, databases, windowing systems, and device drivers are typically purchased today as COTS components, and perhaps even considered part of the hardware platform on which the system is to be built.

In addition to the above component package types, there are others ways of packaging components. Two of interest are the following:

- *Frameworks* as defined in the previous section.
- *OLE Components.* The OLE standard from Microsoft promotes a number of technologies which are intended to enable reusable components. These include OLE Custom Components (OCXs) and OLE Automation servers. At DREV there was a project being undertaken to package a GIS system as a set of OLE component which can be extended through plug-ins from commercial developers.

The successful uses of COTS components which we observed within DND were the following:

- *GIS.* Numerous GIS libraries exist and provide adequate functionality for many applications.
- *GUI builders.* GUI builders are tools which are becoming mature and indispensable.
- *Office automation software.* Software such as calendars, word processors, and spreadsheets are used within many systems. The interaction between these applications and other components of the system were done primarily through desktop features such as drag-and-drop and enclosing files within e-mail messages.
- *E-mail and messaging systems.*
- *Databases.* Databases are an accepted part of many systems and are almost always bought off-the-shelf.
- *Operating systems, including low-level software such as device drivers, windowing systems, etc.*

The above list is made up of applications and components which are mature and pervasive in a large number of systems. This maturity is a likely reason why they have been successfully marketed as COTS software components.

Outside of the mature areas listed above, project and maintenance managers were generally skeptical about the feasibility and benefits of using COTS

components to construct systems. Applications were not generally designed for interworking together, the applications required extensive wrapping, and large amounts of functionality needed to be added to the applications. In the small number of examples where we found COTS applications outside the above domains being integrated, the attitude of the managers was that the exercise could not be considered entirely successful. This was due primarily to problems in integrating and extending the functionality of stand-alone applications that were not designed to be integrated.

The current mechanisms we observed for integrating COTS components were the following:

- Procedure calls. The COTS component is accessed by linking to a procedural interface. Examples include components that are packaged as a procedural library, applications with an API, and databases with an SQL interface.
- Desktop supported capabilities. Desktops provide limited capabilities for integrating components through features such as drag-and-drop, clipboards, cut-and-paste, etc. This is generally how office automation software was integrated.
- Data sharing. For applications that store data in a standard format integration can be accomplished by having components read and write each others data. The shared data can be stored in shared files or in a shared database.

Problems we observed with the integrating COTS components include:

- Stability and support from suppliers was lacking.
- Integrating new releases of COTS software was a labour intensive and error-prone task.
- Contractors were sometimes unfamiliar with COTS product.

#### **4. Experiments with Open Scripting Architecture (OSA)**

Traditionally, commercial software packages have been bought and used on an "as is" basis. Commercial applications are sold as executable packages that have limited functionality for adding new services to the application. Users have been able to install the packages and, except for some minor tailoring, have had to accept the existing functionality of the software application.

Recently, there has been an initiative to open up software applications to allow users and third party developers to extend the functionality of applications and to integrate applications from different vendors to provide more sophisticated services to users. This trend has led to a number of competing and complementary technologies which are intended to allow developers to package open and extensible applications which can be enhanced and integrated by users and third parties.

As part of the research project, we are experimenting with emerging technologies that enable the use of COTS software components. One set of experiments we conducted was to take commercial off-the-shelf applications and to extend their capabilities using the Open Scripting Architecture (OSA). The purpose of this experiment was to determine, from a user's perspective, how effective the OSA currently is for extending application functionality.

The Open Scripting Architecture (OSA) is one of the two scripting architectures competing within the marketplace (the other being OLE automation). OSA is currently under control of the CIL consortium. It was originally developed by Apple Computer and is an integral part of their System 7.5 operating system. OLE automation is part of the OLE standard from Microsoft.

OSA has been adopted as the scripting standard for OpenDoc and thus will be available on most platforms. Currently, OpenDoc, including OSA, has been released for Macintosh System 7, Windows, and OS/2. The architecture is not dependent on a single scripting language. Thus on the Macintosh platform, OSA scripts can be developed in AppleScript, or with some difficulty, Usertalk or tcl. Under OS/2, OSA scripts can be developed in ObjectREXX.

Scripting architectures are designed to allow developers close to the end users (and end users themselves) to integrate and extend applications. They are designed primarily for integrating and tailoring applications rather than for major development work. Applications built according to the OSA standard can extend their behaviour by executing scripts, and can communicate with other elements of the system by exchanging events.

Among the lessons we learned from these experiments are the following:

- Determining behaviour of COTS software components is difficult. Documentation, no matter how well done, is insufficient for understanding the detailed behaviour of components. Other than reading the documentation, other techniques we used for exploring component behaviour included:
  - Experimentation. Understanding the characteristics of a component always required experimentation. The OSA provides a means for CASE tool developers to build environments for experimenting with component behaviour.
  - Examples. Most of the COTS components we integrated had a large user base. Examples and workarounds built up by this user base were invaluable for solving many problems we encountered. This reflects our experience with many popular commercial applications where a large degree of support for the application comes from the user base rather than from the supplier.
  - Browsers. The OSA defines a means by which applications can advertise their objects, data structures, etc. Good browsers help a

developer in categorizing and locating relevant information in an applications interface.

- Recordability. OSA defines a capability whereby events can be recorded and disassembled into an appropriate high-level language. By recording an applications behaviour and studying the program fragment generated we were able to learn about the characteristics of some applications.
- Extending the OSA architecture leads to conflicts between the extensions. The OSA defines a means of adding extensions to the scripting language. Many of these extensions are being developed and marketed by third parties. Unfortunately, with the large number of extensions being developed naming conflicts are arising between the extensions.
- Performance is low. The OSA architecture is currently not viable for time-critical applications.
- Concurrency is not supported.

## 5. Related work

There are a number of initiatives being undertaken by different organizations on using COTS software components. This section lists some of the initiatives which are relevant.

The Software Engineering Institute (SEI) has initiated a project to research the implications of using COTS software to build military systems. Their research has focused on two initiatives: a laboratory project to investigate the integration of COTS CASE tools [ZAR94]; and a two day workshop on the use of COTS in system integration [BRO95]. The workshop proceedings are particularly interesting having brought together industrial and military people to talk about COTS integration within the following frameworks: technical; commercial/business; system architecture; open systems; and acquisition regulations. The workshop identified current problems and future directions of research in COTS integration.

NASA is a strong advocate COTS software and has a number of workshops and initiatives in this area. Their main focus has been on cost and risk reduction through reuse of software components across different projects. Most of their published data has focused on the acquisition/development process needed to foster use of COTS components. They identify a number of successful examples of COTS component usage. In addition to the more common COTS applications of database and user interface tools, they provide other examples such as a mission that integrates three different COTS applications: telemetry and command processing, orbit prediction, and health and safety monitoring [NAS95].

Among the lessons they have learned on successful usage of COTS are the following [NAS95]:

"The following beliefs about COTS packages are questionable

- COTS package solutions are less risky
- I can buy and modify a COTS package more quickly than I can develop it
- There is a COTS package for my application
- The COTS package works because there are a lot of copies in a lot of other organizations
- The vendor will keep the COTS package current.
- Vendor literature is true.

The following beliefs are reality

- Vendors over commit themselves
- Vendors don't supply all services
- The software may not meet the requirements.
- The software may not be easily modified.
- I have very little control over vendor quality and schedule.
- My organization may have to change to accommodate this COTS package.
- Support costs for modifications may be 20% of the cost of the modification per year

Some fundamental differences between COTS based development and custom development are:

- COTS based development may need infrastructure earlier to demonstrate and prove the COTS package
- The COTS package may dictate standards, architecture, and design
- The COTS package may influence work flow
- Picking the wrong COTS package may be more expensive than fixing problems in custom software
- Issue resolution processes need to be in place earlier to resolve COTS package issues
- Issue resolution processes may be more complicated because of the addition of the vendor and possible incompatibilities between the vendor's practices and yours."

A number of new standards are being developed, driven primarily by commercial interests, with the goal of making component based software development a reality. The standards of interest include:

- The CORBA standard [OMG95] from OMG (<http://www.omg.org/>). The OMG, through CORBA, is standardizing an object-oriented approach to re-usable components. The CORBA standard is done at three levels:
  - CORBA which standardizes how component interfaces are specified and method invocations.
  - CORBA services which standardizes services which are common across many applications. Examples of services include security, object life cycle, transaction processing, etc.
  - CORBA facilities which standardizes frameworks (i.e., reusable components) within a specific application domain, e.g., manufacturing, financial services, etc.

The CORBA and CORBA services have been standardized by the OMG. CORBA facilities are being standardized by interest groups within OMG but the standards are not yet in place. Although CORBA compliant Object Request Brokers (ORBs) are available commercially, a market has not yet developed for components built to this standard.

- The Component Object Model (COM) from Microsoft. This provides similar functionality to the CORBA standard (but not CORBA services nor CORBA facilities.) The COM is an implemented standard to which components are being built. It is limited by the fact that in its current form it does not support distribution and works only under Microsoft Windows.
- The document and desktop level component standards of OLE (from Microsoft) and OpenDoc (from CIL). OLE and OpenDoc are currently implemented on a limited number of platforms.

## **6. Issues with COTS software integration**

This section identifies and discusses a number of issues related to building systems from COTS components. The issues discussed are:

- Procurement/development process
- Understanding and evaluating components
- Evolution of software
- Architectural issues
- Role of standards
- Embedded systems

### **6.1 Procurement/development process**

A number of participants commented that one of the key elements to successful use of off-the-shelf software is that to gain the benefits while minimizing the risk an organization must be willing to accept the software component as-is with minimal changes. If the organization is not willing to accept the capabilities and limitations of the software as they exist then many of the benefits of off-the-shelf software will not be realized. This conclusion has been reached not only by ourselves, but by a number of other organizations which have researched the use of OTS software. The Auditor General of Canada which audited a number of large-scale government systems development projects made the following comment while auditing a Transport Canada system that is heavily dependent on COTS software:

"While the acquisition and installation of a commercially available software package should have been the least complex of the systems reviewed, the extensive modification of the software ... significantly increased the project complexity. As well as adding

complexity, such modification may make the implementation of new releases of the software more difficult and costly.... A significant portion of the savings come from limiting the extent of changes to the commercial software package." [AUD95]

Therefore, in order to realize the benefits of COTS software a procurement process must be in place that defines requirements according to what is available in the marketplace, and that is flexible enough to accept COTS solutions when they are proposed.

The issue of impediments to using OTS software in military/government procurements is addressed by NASA [NAS95] and widely discussed in the SEI workshop [BRO95]. Both organizations have emphasized the need to accept the capabilities and limitations of OTS software and to adapt operational requirements to the availability of the COTS components. Currently, rather than approaching procurement with the attitude "what exists off-the-shelf and how can I use it", the procurement process identifies strict requirements which either excludes the use of COTS components, or requires large modifications to COTS packages in order to satisfy the requirements.

"In the old process, system requirements drove capabilities. In the new process, capabilities will drive system requirements...it is not a requirement if you can't afford it"<sup>1</sup>

A number of the procurement issues brought up by SEI workshop include:

- Overly specific requirements often preclude the use of COTS components.
- Use of MIL-SPECS preclude the use of COTS.
- Different development standards are needed for development based on COTS software.
- Allow contractors to submit alternative proposals which may violate some stipulations of the solicitation but may be technically superior due to the use of COTS.

New military standards and procedures [MIL498] recognize the need for developers to include consideration of COTS components during the development process. Paragraph 4.2.3 and Appendix B of the 498 standard provide specific guidelines for incorporating 'reusable software products' into software systems. These guidelines make it clear that unmodified COTS must be handled differently than modified COTS. In general, the standard requires that modified COTS be treated as if it were normal developmental software. This means that it must be fully documented and tested within the scope of the new system. Much of the standalone testing and documentation requirements have been eliminated for unmodified COTS. In this case the standard provides

---

<sup>1</sup> B. Boehm in [BRO95]

for the use of existing documentation and a proven performance record as verification of the COTS software.

## **6.2 Understanding and evaluating components**

A component which is worth buying is likely a complex piece of software. In order to use such a component effectively it is necessary to understand it at quite a deep level. Understanding the behaviour of complex components is an extremely difficult task for a number of reasons:

- The documentation is incomplete or wrong. Even if the OTS supplier is conscientious about documentation, a complex piece of software will not be fully and correctly documented. Even incomplete documentation may be so massive as to be incomprehensible to any but the most experienced users.
- The interface may be very complex. Many of the standard components being marketed (e.g., OLE, DCE) have APIs with hundreds of calls. Not even the people working on these systems know how each API call behaves or the effect of particular sequences of calls.
- There are bugs in the software.

There is no easy solution to understanding complex systems. Documentation must be available and used, but a deep understanding can only be gained through extensive experimentation. Trying to understand these components has been called "an experimental science without any laws of physics." One advantage of using a COTS application with a large user base is that much of the experimentation has been carried out and the knowledge is available in the user community.

Many of the problems encountered with integrating COTS components cannot be determined before integration begins. The problems are of such a nature that they only appear well into the integration process [GAR95]. This makes estimation of schedules and resource requirements extremely difficult where COTS components are used. Extensive evaluation of the COTS component will be required to ensure not only that the component has the functionality to perform the required tasks within the system, but also that the additional functionality inherent within the component does not interfere with the system. The cost estimates must take this evaluation process into account.

## **6.3 Evolution of software**

Systems are constantly changing; COTS components within systems are constantly changing. This evolution of systems and their components has an impact in a number of ways on the maintenance of systems.

All people we interviewed stated that there was a large amount of work required to integrate new releases of COTS software into their systems. For a system constructed from multiple COTS components, with each component having its

own release schedule, the cost of integrating each new release of each component becomes prohibitive.

The difficulties arise for two reasons:

- In the new release of the COTS component there will likely be changes in the behaviour, interface, assumptions, performance, bug fixes etc.
- Specializations, extensions, and workarounds made to the older version of the COTS component which must be integrated into the new component. There is no concept of "compliance" as in mechanical systems where parts are not defined to fit exactly but rather within certain tolerances.

One approach to the problem of evolution is to assume that the system (or at least the OTS component within the system) will never be updated. This will not be practical in all cases. The software may be dependent on a particular platform that is no longer available or is being updated; operational requirements may change; or bug fixes in the new releases may be required.

Integrating a new release of COTS software requires a significant development effort. First, the new release of the COTS software must be evaluated to determine what has changed from the previous release. Second, the new component must be integrated into the system. This may involve adding or removing workarounds, adding new extensions to take account of new behaviour, updating documentation and training procedures, etc. Finally, the system must be tested and verified.

#### **6.4 Architectural issues**

An appropriate software architecture addresses three issues related to the use of COTS software components:

- Plug-and-play of components. An architecture should allow pre-built components to be quickly assembled into larger systems. Components should be replaceable with a minimum amount of effort.
- Sharing of components/knowledge across projects. Within DND, many projects are implemented using a "stovepipe" model where each project develops components through all layers of the system with very little re-use or sharing with other projects. Many projects within DND have similar functionality and requirements; an appropriate architecture would encourage sharing of components across projects to take advantage of this shared functionality.
- Building systems as components for larger systems. All systems built should assume that one day they will be the off-the-shelf component which will be integrated into a larger system. Systems currently being built within DND have limited capability to be integrated into other systems. While this is a highly desirable trait, it is obvious that commercial vendors will be reluctant to add increased complexity to their product (at increased development cost)

to provide interworking with other systems unless it can be shown that there will be a return on investment. If we assume that DND cannot effectively influence commercial vendors, then the viewpoint should shift to ensuring that the COTS "glue" is flexible enough to accommodate integration with future systems.

## **6.5 Role of standards**

There is clearly a role for standards in enabling the use of COTS software for building systems. If standards are defined correctly and if software developers build components which comply with the standards then this goes a long way towards making open systems a reality. There are already a large number of standards which have had an impact in building plug compatible software components. Examples of these include tcp/ip, Xwindows, and SQL. There are many other standards in many other domains that have the potential to have a similar impact. Some of these address directly the issue of interoperable components (e.g., OLE, CORBA, ODP, OpenDoc, etc.) while others are targeted at a specific functional domain.

Although standards are important it must be recognized that there are many problems that standards do not address. For example, regardless of how pervasive standards become, they do not in any way eliminate the need for evaluating and experimenting with COTS components or solve the problems associated with maintenance and evolution of systems containing COTS components.

Another issue that must be recognized by DND is that to gain the benefits of standards it must be willing to accept the de facto standards which have gained acceptance in the marketplace. DND does not have the market force to dictate standards and attempts to mandate standards will likely lead to the use of software that is not widely supported in the commercial world. The role of DND should be to monitor and understand the standards, and possibly to try and influence the standards definition process, but once a standard has been accepted in the commercial world, be it a de jure or a de facto standard, this standard must be accepted as is by DND.

## **6.6 Embedded systems**

Although using COTS components is usually associated with information systems there are instances where COTS components can be effectively applied to real-time and embedded systems. There are many examples of real-time systems which make some use of COTS components. At the SEI workshop, the claim was made that the Boeing 777 has 4 million lines of COTS software including Microsoft products. (It was also claimed that to Bill Gates, Boeing was too small a customer to meet with.)<sup>2</sup>

---

<sup>2</sup> Stated by Claude M. Del Fosse, SPC, in [BRO95]

Areas where COTS software components do have impact on embedded systems include the following:

- Embedded software. When buying a specialized piece of COTS hardware there will almost always be software embedded in the equipment.
- Specialized expertise. If embedded systems require access to highly specialized expertise, this expertise may be more readily provided by COTS software rather than attempting to hire the appropriate experts.
- Operating systems, etc. Embedded systems will generally use commercial operating systems.
- Standard non-critical components. Many embedded systems will continue to contain standard components that can be bought off-the-shelf and integrated. These include for example report generators, GUIs, databases, etc.

Issues to be resolved include:

- Certifying components that are critical. If a COTS component is to be embedded in a critical system, a rigorous certification process must be performed on the component.
- Constructing firewalls between the critical system and uncertified COTS components.

## **7. Future research**

Possible future directions of research relating to COTS software integration include the following:

- Develop techniques for evaluating and experimenting with COTS components. Select a number of products and attempt to define or develop tools which could be applied to a COTS component to pre-determine the overall functionality of the component. This may lead to the definition of a classification system which would allow project groups to quickly create short-lists of appropriate software components for their particular system. Another approach could be to attempt to analyze and establish evaluation criteria to be used to determine the suitability of a COTS component within the framework of a specific system.
- Define and evaluate technologies and architectures that promote reuse across projects and allow applications to be integrated into larger systems. Issues which could be investigated include:
  - Coordination languages designed specifically for integrating and extending COTS applications.
  - Component architectures, including scripting architectures and frameworks.

- Architectures which facilitate component reuse across projects.
- Experimentation and monitoring of standards (CORBA, OLE, etc. ...). Experimentation is required to fully understand the functionality of these standards and determine how they may (or may not) facilitate the use of COTS software.
- Develop a guidebook for project and maintenance managers that identifies issues associated with COTS development. Contents of the guidebook could include:
  - Overview of development process applicable when integrating COTS components.
  - Costs, benefits, and risks associated with COTS software reuse.
  - Lessons learned from previous experiences with COTS components.
  - Tools and techniques for integrating components.
- Pick an application domain within DND and determine how current state-of-the-art commercial software satisfies the needs of DND and how DND might change its systems to take advantage of COTS within this domain.

1/25/96

- [NAS95] NASA workshops:  
<http://bolero.gsfc.nasa.gov/c600/workshops/sswssp4b.htm>  
[http://www-isds.jpl.nasa.gov/isds/cwo's/cwo\\_23/pbd.htm](http://www-isds.jpl.nasa.gov/isds/cwo's/cwo_23/pbd.htm)
- [ZAR94] P.F. Zarrella, A.W. Brown, "Replacing the Message Service Component in an Integration Framework", SEI Technical Report CMU/SEI-94-TR-17, Software Engineering Institute, Pittsburgh PA, 1994. <http://www.sei.cmu.edu/products/publications/94.reports/94.tr.017.html>
- [BRO95] "Proceedings of the SEI/MCC Symposium on the Use of COTS in Systems Integration", A.W. Brow, D.J. Carney, M.D. McFalls ed., SEI Special Report CMU/SEI-95-SR-007, Software Engineering Institute, Pittsburgh PA, 1995.  
<http://www.sei.cmu.edu/products/publications/95.reports/95.sr.007.html>
- [MIL498] MIL-STD-498, "Software Development and Documentation"  
<http://sw-eng.falls-church.va.us/AdaIC/standards/mil-std-498/>
- [OMG95] "The Common Object Request Broker: Architecture and Specification, Revision 2.0", Object Management Group, Framingham, MA, 1995.
- [AUD95] "Report of the Auditor General of Canada to the House of Commons. Chapter 12, Systems under Development: Managing the Risks", October 1995, Supply and Services Canada.
- [GAR95] D. Garlan, R. Allen, J. Ockerbloom, "Architectural Mismatch or Why It's Hard to Build Systems out of Existing Parts", in 17th International Conference on Software Engineering, pp179-185, ACM, 1995.