Maintaining a COTS-Based Systems

Vigder, Mark; Dean, Joh

Proceedings of the NATO Information Systems Technology Panel Symposium on Commercial Off-the-Shelf Products in Defence Applications, 2000

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de Technologie de l'information

NRC-CNRC

*Maintaining a COTS-Based Systems**

M.R. Vigder and J. Dean
April 2000

***published in** Proceedings of The NATO Information Systems Technology Panel Symposium on Commercial Off-the-shelf Products in Defence Applications, Brussels, Belgium. April 3-5, 2000. 6 pages. NRC 43626

Maintening a COTS-Based Systems

Dr. Mark R. Vigder
John Dean
National Research Council of Canada
Institute for Information Technology
Ottawa, Ontario
Canada
K1A 0R6
{mark.vigder|john.dean}@nrc.ca

Summary: After depoyment, all software systems require an extensive and expensive phase of maintenance and management regardless of whether they are COTS-based or custom built. Understanding how COTS-based systems are maintained, and why they are different from custom built systems, can lead to systems that are better and more cost-effective over their lifetime.

1 Introduction

After deployment software systems enter a phase of maintenance, management, and evolution that can last many years until final decommissioning [3,5]. This post-deployment phase is the longest and hence the most expensive phase of the software lifecycle. Success during this phase is often the determining factor as to whether a software system is cost-effective over its lifetime.

Building a software system from COTS products does not change the importance nor the expense associated with maintenance, evolution and management. COTS-based systems must continue to satisfy evolving user requirements, failures of the system must be dealt with, the system must adapt to the ever-changing environment, and managers must be able to monitor and control the deployed system. These activities are necessary whether a system is built from scratch or built using commercial products.

The nature of the post-deployment activities changes when dealing with COTS-based systems rather than with custom built systems. If COTS-based systems are to be successful over the many years that they are expected to be in service, organizations involved in building or acquiring COTS-based systems must understand and accommodate these differences.

2 COTS-based systems: why is maintenance different?

Software maintenance includes all the activities required to evolve a software system over its lifetime. Although the motivation for maintaining COTS based and custom systems is the same, the nature of the activities required of the maintenance personnel is different. The different activities required for maintaining COTS intensive systems arise for a number of reasons.

Primary among the reasons for the different maintenance activities is the fact that the evolution and upgrades for the individual COTS products are outside the direct control of the system developers and acquisition organizations. The COTS products are maintained and supported by the COTS product developer or their agent. The system developer must treat these products as single black-box entities with little or no visibilty into the internals of the product and perform the maintenance at the level of large-scale products rather than at the source code level. The only source code being maintained by the system developer is that required for integrating the large-scale COTS products. This includes code for wrapping and tailoring the individual products, as well as the “glue code” required to connect the products together. Wrapping and tailoring of the products (without accessing the products source code) becomes necessary to

overcome architectural mismatch between products, to customize the product to conform to local requirements, and to build workarounds to overcome the inevitable bugs (and features) that are included in any COTS product.

From the acquisition agencies' perspective, they have effectively ceded control over maintenance and evolution of large parts of the system to outside commercial agencies. Maintenance of the COTS intensive system is now driven in a large part by the vendors of the different products rather than by the system developer. In effect, having amortized the cost of development and maintenance among many different users, acquisition agencies are now one among many users driving the direction of the COTS software evolution.

2.1 Maintaining a COTS-based system

In order to more effectively maintain and manage COTS-based systems it is necessary to identify the activities of the maintenance and management personnel. Once the activities have been identified, strategies can be developed to facilitate these activities. COTS-based maintenance and management, although similar in many respects to maintaining custom-built systems, has qualitative differences. These differences result in the following activities in the post-deployment phase (Table 1).

Component reconfiguration. Reconfiguring components is the act of replacing, adding and deleting components within the system. Reconfiguration occurs for many reasons, perhaps the most common being the frequency with which commercial product vendors release updated versions of their software. It is not uncommon for each product to be upgraded two or three times per year. Often, system integrators are forced to replace older product versions with the upgrades in order to fix bugs or improve functionality. Other reasons for reconfiguring the components are to replace aging components with better products from competing vendors, or to add and delete products as the functional requirements of the system evolves.

Reconfiguring the components is an expensive activity requiring the integrators to go through a complete release cycle including product evaluation, testing, design, integration, and system regression testing.

Troubleshooting and repair. All systems fail and COTS-based systems are no different in this respect. However, with COTS-based systems maintenance and management personnel generally cannot look inside components when trying to isolate the cause of the failure. Information must be gathered by experimenting at the edges of the components. Identifying the source of the fault requires running a series of experiments to determine the product or products causing the problem [2].

Identifying and fixing the fault is no longer an activity performed solely by the system builders. Having used third-party products, system builders must now work closely with the support staff of the product suppliers, and with the general product user community. Where faults involve complex interactions involving sets of products from different vendors, many different organizations may be involved in the troubleshooting and repair of the system.

Configuration management. For COTS-based systems configuration management is done at the level of products rather than at the level of source code. Issues that maintainers must address include: change history for each individual product; availability and support level provided by the product vendor; management of configurations of the COTS-based system that are installed at each deployed site; compatibility requirements and constraints between sets of products; and licensing issues associated with each product.

Testing and evaluation. Testing and evaluating COTS products is an ongoing activity during maintenance. New product

Maintenance activity	Description
Component reconfiguration	Updating product versions, replacing COTS products with similar products, adding/deleting products
Troubleshooting	Identifying causes of failures among sets of COTS products, developing workarounds with the products, liasing with the COTS product maintainers
Configuration management	Tracking versions of different COTS products, tracking deployment configurations, determining compatible versions of products
Testing and evaluation	Testing new product versions as they become available, within the context of the system into which they will be integrated
Tailoring user level services	Enhancing the services available to the end user by configuring COTS products, combining services of multiple products, etc.
System monitoring	Monitoring different aspects of system behaviour, such as communication, resource usage, process invocation, etc.

Table 1. Maintenance/management activities for COTS-based systems.

versions as well as new products must be evaluated for includion within the system and products must be tested during operational use.

Tailoring user level services. COTS products provide a generic functionality that can be used by many applications and organizations. System integrators must customize and tailor this functionality to satisfy the local operational requirements that are unique to the end-user organization. Successful systems are those that can be quickly modified and tailored to meet evolving user requirements.

For COTS-based systems tailoring involves an ongoing process of customizing and configuring products, adding new components to the system, and combining services of multiple products in novel ways. Since integrators do not have access to product source code this must be done through gluing products together to provide enhanced functionality and using vendor supported tailoring techniques to customize the products.

System monitoring. System managers and maintainers must continuously monitor a system during its ongoing operation. This must be done to measure performance and resource usage, watch for failures, and determine user behavior. Because COTS software is black box, with limited visibility into internal behavior, monitoring for maintenance purposes can be difficult to do effectively.

3 Planning for post-deployment

Systems are maintainable and evolvable through their lifetimes because they were explicitly designed to be so. Maintainability cannot be built in “after the fact” but must be considered during the early stages of analysis and design.

For COTS-based systems, there are two phases of construction during which system builders must consider maintainability and evolvability. The first is during product evaluation and selection. The products used to build the system have a great deal of impact on the characteristics of the system during its maintenance.

The second phase of construction that impacts system maintenance and evolution is the architecture and design of the system. Different architectural styles have different properties relative to the evolvability and maintainability of a system. By identifying the properties required of a COTS-based

system an appropriate architectural style can be applied that provides these properties.

3.1 Product selection

System builders do not control the individual products, but they do control which products are selected for integration into the system. There are many different criteria used for product selection but system evolution should be one of the factors considered when developing criteria for product selection. A number of properties of a product affect the long-term evolution and maintenance of the system.

Openness of the component. A component is open if it is designed to be visible, extendible and easily integrated into a wide array of systems. In general, the more open a component the easier it will be for maintainers and managers to monitor, manage, extend, replace, test, and integrate. Many factors combine to make a component open and among things to consider are: adherence to standards; availability of source code perhaps through open source licensing; and ability to interwork with products from many different vendors.

Tailorability of the product. Much of the maintenance effort for COTS-based systems involves tailoring the functionality to meet evolving user requirements. One of the criteria for product selection should be the ease with which the product can be tailored to satisfy local requirements. Although products are black box and the source code cannot be modified there are many techniques product builders can use to make their products tailorable. Examples of tailoring techniques include scripting interfaces, data configuration files, and frameworks that can be extended through the use of inheritance and plug-ins.

Available support community. System builders require extensive assistance from external organizations to support commercial software. This support comes from the vendor and the user community. Given that successful maintenance is dependent on this support, system builders must evaluate the support available for the product during the product evaluation process.

3.2 Designing for evolution

System builders do not own COTS software, but they do own the architecture and design used to integrate the software. By addressing issues of maintainability during the software design activity, designers can build a system that facilitates the maintenance activities associated with COTS-based systems and avoids may of the pitfalls [1].

There are two major issues that can be addressed when designing COTS-based systems for maintainability. The first is the management of dependencies between the diverse software elements. Many uncontrolled dependencies between products make it exceedingly difficult to modify or analyze a software system. Component replacement or addition will be difficult due to the affects that can ripple through the various component dependencies. Many intedependencies also make understanding failures and isolating faults a more complex task.

The second design issue that must be addressed is visibility into the system. A system is visible if maintenance and management personnel can instrument and monitor the system. This involves querying the system to determine its operational characteristics, current configuration, fault incidents, etc. Visibility is a necessary characteristic for testing and managing systems. For COTS-based systems, where there may be limited visibility into the individual products, designers must build visibility into the architecture.

3.2.1 Managing product dependencies

Complex and intricate product dependencies result in a fragile system in which it is difficult to upgrade, replace, add and remove components. To alleviate these problems, designers must manage the dependencies

Architectural view	Entities
Interconnection topology	Map of the data flow between the COTS components.
Connection infrastructure	Mechanism used to transfer data and control among the software elements, e.g., CORBA, DCOM, RMI.
Interfaces	Exposed parts of the COTS software product.
Collaborations	Ongoing behaviour required among a set of components in order to provide a service.
Environment	Dependencies made by the COTS product about the environment in which they are operating, e.g., operating system, software versions, file structure, etc.
Control mechanism	Dependencies caused by assumptions about process structure, control flow, activation, etc.

Table 2. Causes of component dependencies.

between the products so that COTS-base maintenance is possible.

There are many ways that software components within a system may be dependent. Some of these are explicit, such as the direct transfer of data through an exposed interface. Other dependencies are implicit and subtle, such as conflicting assumptions that different software products can make regarding the environment under which they are executing.

Table 2 summarizes the major causes of component dependencies. It is important for designers to recognize that they cannot eliminate dependencies, but they can manage them in a way that allows for more effective maintenance [4].

3.2.2 Designing for visibility

Visibility is a property of a system that permits inspection and instrumentation by managers and maintainers. Capabilities involved include event logging, raising alarms, determining the current configuration, etc. Visibility is necessary for debugging, testing, isolating faults and managing the system.

Designers have little or no control over the visibility provided by the individual software products. However, through the architecture and design a great deal of visibility can be built into the system by using the glue and integration code as tools for monitoring and viewing the system. An example is shown in Figure 4 in which a mediator exposes a management interface. Through this interface different information about the collaboration and the components can be gathered such as the events generated and received by the components, activiations of the components, component versions, etc.

4 Conclusions

Although component-based software systems provide many advantages, designers and users must still expect that the majority of the lifecycle cost will be incurred after the initial deployment of the system. Reducing this cost, and easing the maintenance and management effort, requires designers and architects to consider the post-deployment activities during the earliest stages of software development. By identifying the activities that maintenance and management personnel perform to support component-based systems, and using a design that supports these activities, systems can be made more cost-effective.

Bibliography

[1] David Garlan and Robert Allen and John Ockerbloom. Architectural Mismatch or Why it's hard to build

systems out of existing parts. In 17th International Conference on Software Engineering, pp179-185 1995.

[2] Scott Hissam. Correcting System Failure in a COTS Information System. In Proceedings of the International Conference on Software Maintenance, pp170-176, Nov 1998.

[3] Duane W. Hybertson and Anh D. Ta and William M. Thomas. Maintenance of COTS-Intensive Software Systems. Journal of Software Maintenance, 9(4):203-216, 1997.

[4] Mark Vigder and John Dean. Building Maintainable COTS-Based Systems. In International Conference on Software Maintenance, pp132-138, 1998.

[4] Mark Vigder. The maintenance, management, and evolution of component-based systems. In *Component-Based Software Engineering: putting the pieces together*. Adison-Wesley, to be published, 2000.

[5] Jeffrey Voas. Disposable Information Systems: The Future of Software Maintenance?. Journal of Software Maintenance: Research and Practice, 11:143-150, 1999.