Building Maintainable Component-Based Systems

Dr. Mark R. Vigder
National Research Council Canada
mark.vigder@nrc.ca

Abstract

Maintaining large software systems that are constructed from pre-built components is expensive and one of the major cost drivers for these systems. By identifying the activities associated with maintaining component-based systems, and then designing systems that facilitate these activities, maintenance costs can be reduced. Designing maintainable systems requires a specific set of design criteria that are to be followed, and a checklist of items that can be used during design inspections in order to verify that the design criteria are being adhered to.

1. Introduction

The traditional approach to a discussion of maintenance issues focuses on the ideas of adaptive, corrective, preventative and perfective maintenance. These discussions, and any conclusions reached, assume that the maintainer has full access to the source code from which the system is constructed. However, many modern software systems are a built from software components where these components are a combination of custom built software and Commercial Off-the-shelf (COTS) software that are acquired and evolved over a long period of time. With the use of third-party components the system manager does not have access to source code (often a blessing) nor direct access to the developers.

Maintenance of these software systems has become a significant cost driver in an organization's overall IT costs. Many of the maintenance activities are made more difficult due to the use of components. Maintaining the system involves managing a number of black-boxes that are supplied by many different vendors that were not necessarily designed to work together.

Maintenance of component-based systems differs from maintenance of custom-built systems in the following ways:

System developers do not have access to the source code. Without access to source code, maintainers must find novel ways for troubleshooting and supporting a system, test the system, and change and modify the functionality of a system.

Maintenance and evolution of the component is controlled by a third party. System developers are one more customer of the component developer. It is the component owners who control the evolution and maintenance of the individual components.

Maintenance is done at the component level rather than the source code level. Rather than fixing

components and modules, maintenance of component-based systems involves replacing/adding/deleting components rather than source code changes.

This position paper discusses three main issues related to maintenance: Section 2 identifies the major maintenance activities of component-based systems; Section 3 describes architectural properties that can be built into a system that support the maintenance activities; and Section 4 outlines how an inspection checklist can be used during system construction to assist in building a maintainable component based system.

2. Component-based maintenance activities

The activities of the maintenance process for component-based systems are activities associated with the component level rather than with source code level. These activities include the following.

Gluing and wrapping. Components are generally not "plug-and-play". Significant effort is required to build wrappers around components and the "glue" between components in order that they can work together. As systems evolve this wrapper and glue code must be maintained.

Tailoring. Components provide generic functionality. Organizations must tailor this generic functionality to correspond to their unique business requirements. Such tailoring can be done through various technologies without touching the source code of component, e.g., scripting, plug-ins, and frameworks. As businesses modify and update their processes, maintenance of the software system must be performed to reflect the modified business processes.

Fault identification and isolation. When a system fails, system maintainers can no longer fix the problem by changing the source code. Fixing the code is the responsibility of the component builders. The maintainers must deal with the failure by identifying the component (or set of components) that have the fault, and then by isolating the fault and finding workarounds in order to continue using the system.

Updating component configuration. One of the major activities of maintaining component-based systems is the effort required to upgrade component configurations. This includes: replacing components with newer versions as they are released by the component developer; substituting similar components from different vendors; adding or deleting components as the requirements of the system change.

Monitoring and auditing system behaviour. Any system must be monitored in an ongoing manner in order to evaluate issues such as performance, process improvements, failure detection, and usage as well as to assist in troubleshooting. For component based systems this requires a system manager to be able to monitor the load on each component, its failures, component performance, how components are being used, and where they are being used.

Component testing. System maintainers are continually adding and upgrading components within a system. Before adding components, system maintainers are required to perform extensive component testing to determine the effect of integrating the component into the system. The testing must be done to determine behaviour of the component, differences from

previous version, performance, resource requirements, etc.

3. Software architectures that facilitate maintenance activities

Overall system lifecycle costs of component based systems can be reduced by building systems in such a way that they facilitate the maintenance activities that were outlined in Section 2. By building a system with appropriate architectural properties, maintainers will have techniques available to perform the activities. Following is a summary of architectural properties that should be built into component-based systems to enhance their maintainability.

Service level tuning. Maintenance personnel must be able to easily modify the user services of the system. As an organization fine tunes its business process, it must update the software that embodies the process, i.e., maintainers must be able to tune the services that the system provides. Since components cannot generally be modified directly (and should not be modified) it is the responsibility of system designers to build component-based systems where the user services can be tuned without component modification. This can be done through the selection of components that allow for tuning (e.g., through scripting or plug-ins) or by a software architecture that separates generic functionality provided by components from the specific business process functionality that can be provided by modifiable mediators or scripts.

Flexible component configuration. A flexible component configuration refers to the ease with which maintainers can add, delete, replace, and upgrade components. Factors affecting the flexibility include the level of component coupling, interconnection architecture among components, and the use of wrappers and glue code to isolate components.

Visibility. In order to assist in fault identification and isolation, troubleshooting, and system monitoring, it is necessary that system maintainers have visibility into system behaviour. Since components themselves are black boxes, visibility must be provided at the edges of the boxes, in the wrappers and glue code. Developers must build systems in such a way that monitoring and instrumentation facilities are provided at appropriate points in the architecture.

Fault isolation and exception handling. A software architect cannot control when or how a component will fail, but they can control how failures are detected and what happens in the event of a failure. For maintenance purposes, it is important to design systems such that failures are detected as soon as they occur, are isolated to the components in which they occur, an exception is generated for every failure, and recovery and reporting is handled by the system architecture.

Open system. An open system, in this context, is one that can be expanded and included as part of a larger system. Openness is achieved through a number of means, for example use of standards for interfaces, standards for middleware, exported and documented interfaces and data schemas, etc.

Appropriate integration architecture. System builders do not have ownership of the components of the system; they do however have ownership of the system architecture. Architects must focus on constructing a system that assists integration by: minimizing component coupling; allowing for concurrency control; and provide instrumentation capability.

4. Constructing maintainable component based systems

In order to build a system with the properties outlined in Section 3, a mechanism must be put in place that verifies the existence of the properties during the construction and evolution of the system. One mechanism that can be used to verify these properties is to identify a specific checklist of items that can be verified in the design and implementation of the system. A checklist can be used during the inspections or walkthroughs of the software design; if the inspection verifies the items on the checklist are included then the system will possess the desired properties.

Categories that should be included as part of the inspection are listed below. A more detailed description can be found in [1,2].

Connector infrastructure. Since connectors provide the infrastructure for communication between components, they impact the ease of integration and operations of the system. The objectives in inspecting the connector architecture are: determine whether components can be easily moved to allow for reconfiguration of the system; verify that components can be easily added, removed, and substituted; maximize the selection of third party component suppliers who can provide components that are compatible with the system; determine the level of visibility the connector provides for the purposes of troubleshooting and testing.

Interconnection topology. The objective of evaluating the interconnection topology is to minimize the number of interconnections and to simplify the interconnection patterns. Understanding the topology is a necessary condition for understanding component dependencies and performing operations such as developing test plans and system troubleshooting. Simplifying the topology eases the task of substituting components and integrating new components into the system. It also provides an insulating mechanism between components by isolating functionality.

Interfaces. The objective of evaluating the interfaces is to verify that interfaces provide the following: interfaces remain independent of the underlying component so that component substitution is possible; interfaces facilitate the instrumentation, monitoring, testing, and troubleshooting of systems; and interfaces are documented, versioned, and under configuration management.

Tailorability. The objective is to have a system that can be quickly and easily tailored to meet new or updated requirements. A checklist must verify: the tailorability of individual components; tailorability of the glue code; and tailorability of the architectural style.

Architectural style. The objective of the architectural style evaluation is to verify that: an architectural style for the system has been defined and documented; the style is appropriate for the long-term evolutionary goals of the system; components used to construct the system are consistent with the architectural style; the architectural style is being maintained during the initial development as well as during the maintenance of the system.

Run-time instrumentation. The purpose of evaluating the instrumentation is the following: determine whether integrators and users have an adequate instrumentation capability; determine

what level of instrumentation is available from the architecture; determine the usefulness of standards for instrumentation.

Collaborations. The objective in inspecting the component collaborations is to determine the dependencies between components and to verify that system services can be added and modified.

Configuration management. Since much of the maintainability of component based systems depends on an appropriate configuration management process to manage the evolution and configuration of components, it is important to evaluate the CM processes and tools. It must be verified that the CM process can: determine the component versions installed at each site; track the history of updates to components at each deployed site; and record known compatible and incompatible sets of components.

Component substitution Evaluation of the architecture should include a number of component substitution exercises to verify that component substitution is possible, and determine the level of effort required to substitute components.

5. Conclusions

Maintenance activities associated with component based systems are different from traditional systems in that maintainers are dealing with components that are black-box, they do not control, and which they exercise minimal control over how they evolve. In order to reduce life cycle costs, architects must design the system in such a way to facilitate the activities associated with maintaining component based systems. This can be done by identifying the architectural properties that are desirable in such a system and then developing a detailed checklist that can be used during inspections and/or walkthroughs to verify that the properties are being built and preserved in the system.

[1] M. Vigder, *Inspecting COTS Based Software Systems, Verifying an Architecture to Support Management of Long-Lived Systems*, NRC Report No. 41604, 1998. (Available at http://wwwsel.iit.nrc.ca/projects/cots/COTSpg.html).

[2] M. Vigder, *An Architecture for COTS Based Software Systems*, NRC Report No. 41603, 1998. (Available at http://wwwsel.iit.nrc.ca/projects/cots/COTSpg.html).