

APPLICATION ARCHITECTURE ANALYSIS

ASSESSING EXISTING SYSTEMS

PRINTED: TUESDAY, MAY 24, 2005

© HILBERT COMPUTING, INC., 2003

APPLICATION ARCHITECTURE ANALYSIS

ASSESSING EXISTING SYSTEMS

TABLE OF CONTENTS

EXECUTIVE OVERVIEW.....	2
WHAT IS THE ARCHITECTURE?.....	3
ASSESSING COUPLING AND BINDING.....	6
ASSESSING SUPPORTABILITY.....	8
CONCLUSION.....	10

EXECUTIVE OVERVIEW

There are many reasons to assess an existing software application. The business environment may have changed significantly since the application was first designed to due shifting market conditions or due to regulatory changes. Mergers and acquisitions also require that companies assess the state of applications providing similar business functions to determine which applications become used in the reorganized business. There may be software that was introduced onto the market that triggers a buy versus maintain decision. The key to the decision, obviously, is determining the cost of ownership for the remainder of that software's lifecycle.

One of the most effective ways to manage the total cost of software is to ensure that it has a strong architectural basis. A strong architectural foundation makes an application more resilient to change because portions of the application can be redesigned and reimplemented without a “ripple effect” of those changes throughout the application. The assessment of the architectural basis for the software may be as important as the specific business functions it provides. In almost all cases, the software will go through updates to respond to changing requirements, so the current implemented business functionality will likely play a less significant role in the total cost for the remainder of the application lifecycle than its ability to adapt without significant portions being rewritten.

Applications built with a strong underlying architecture are more likely to contain software elements that aid in the diagnosis of problems while the application is in production. These operational aspects of an application are often overlooked in favor of the functional requirements of the application, but are important not only for lowering the support costs, but in terms of availability and customer satisfaction.

It is my recommendation that when making a choice of which code base to maintain, an architectural analysis will be the key to making the most cost-effective decision for your organization.

WHAT IS THE ARCHITECTURE¹?

The inherent power of software is that it can take on any shape. There are countless ways to solve any given business problem in software. There are also countless ways in which that software can be structured. The application architecture defines the basic “shape” of the application. It contains the structural elements for separating the user interface from the business logic and the data access portions. The architecture also defines the structure of the software elements that define the flow of the application. While the architecture does not define the way in which the software parts are coupled together, it defines the range of possibilities in which coupling will occur.

When making an assessment of the application architecture, the architecture must first be discovered, or a determination must be made that there isn't a coherent application architecture. In the ideal situation an application architecture is readily apparent through the use of a “framework”. A framework is a collection of software that is independent of a business problem domain, whose implementation is the manifestation of the architecture. That is, the architecture for an application may be driven by deployment issues such as scalability or accessibility², but it is not driven by the details of the business process being implemented within an application. The architecture may define a model-view-controller design pattern for the management of the workflow within the application. If that is the case, as it often is, then the framework would implement the controller and the ways in which the model and view layers interact with the controller to make requests of each other.

The application architecture may also define the possibilities for interacting with the user and interacting with events external to the application. The most tangible of these is the user interface technologies that are supported within the architecture, such as full-function browser-based web applications, micro-browser found in mobile devices or rich-client user interfaces found in desktop resident applica-

-
- 1 Really, there are several architectures within an enterprise. For example, there should be an application architecture, data architecture, a security architecture, operational architecture, etc. Since this discussion is focused on application architecture, we imply “the” architecture to be the application architecture.
 - 2 Accessibility may drive architectural decision such as whether it is a web-based application or a desktop application. If it needs to be globally-accessible, web-based is the most likely choice.

APPLICATION ARCHITECTURE ANALYSIS

ASSESSING EXISTING SYSTEMS

tions. That is, there may be a separate application architecture within the enterprise for each of the broad deployment models.

The application architecture also defines the process and threading model within the application. A process³ is an operating system term that defines a single addressable range of memory. Within the process, there are separately dispatchable units of work that can run concurrently. Each of these is a thread.

Some containment environments, such as some of the J2EE⁴ specifications define the process and thread model as part of the specification. A servlet will allow multiple applications within a process, just as CICS does on mainframe systems. For servlets, the threading model is mostly single-threaded from an application programmer's point-of-view, but multiple requests can be run in parallel since the container provides a pool of threads on which to run browser-originated requests. More complex containment environments, such as J2EE's EJB⁵, allow a single application to be distributed among multiple processes on multiple machines.

For non-container environments, there is more choice for threading models. The programmer may choose to implement multiple asynchronous activities within a single application running within a single process.

An assessment of the threading model can assist in the internal complexity of the application and therefore the costs to maintain and extend the application. The simplest threading model is single-threaded, of course, but if the application requires parallel activities and the architecture doesn't accommodate that need, the workarounds⁶ can be a significant source of code defects. If the architecture supports multi-threaded applications and the threading is exposed to the applications developer, then there is a possibility for added complexity. As a rule-of-thumb, multi-threaded applications should be built on a framework developed by a

3 "Process" is the most common term used for this operating system concept. Those of us with a mainframe background may call this an "address space" or "partition".

4 J2EE stands for Java 2 Enterprise Edition and defines several specifications, the most common of which is Java servlets

5 EJB stands for "Enterprise JavaBeans". It provides programming interfaces for distributed applications with roots in CORBA as well as different persistence mechanisms. It is somewhat complex and heavyweight, but provides powerful features for those applications that require it

6 I usually just call them "hacks".

APPLICATION ARCHITECTURE ANALYSIS

ASSESSING EXISTING SYSTEMS

knowledgeable architect or very senior developer or the supportability of the application can be exceptionally expensive.

ASSESSING COUPLING AND BINDING

The ability of software to adapt to change is directly related to two concepts: coupling and binding. To understand coupling, we need to look at the opposite - *decoupling* - to gain an appreciation for why decoupling is so important for applications to be responsive to change with a short time-to-market.

Decoupling parts of an application means that those parts have little or no dependencies on each other. That allows change to occur in one part of the application without having an affect on another part of the application. In the procedural programming days, a change to a data structure could have a ripple-effect throughout the application because every module that had access to that data structure may change as a result of a change to the data structure. Object-orienting programming languages such as Java have the potential to reduce the coupling through the use of interfaces and the use of design patterns to separate the concrete implementation from the basic behaviors defined in the interface⁷.

Decoupling components within an application also aids in shortening the time-to-market for new features to be added to an application. Because the parts of the application are decoupled, new features can be implemented with a high degree of confidence that they won't affect existing functionality. The concept is simple: if my code can't touch the other code within the application, my code can't break it.

A good application architecture is defined at a fairly high level of abstraction. There must be sufficient structure so that the application code can be supported by the architecture, but there shouldn't be too much structure such that implementation possibilities are overly constrained⁸. The *binding* within an application is where the rubber meets the road. In short, the architecture defines the key abstractions and the bindings define how those abstractions come together to deliver specific, tangible business functionality. In procedural programming, binding often occurred at the source-code level. Copybooks in COBOL defined the concrete data structures available to the parts of the application. The effective use of function libraries and partitioning of the procedural code could push that into link time binding, but most procedural code is bound to the other procedural code within the application very early in the development process.

7 the factory design pattern is used most often to accomplish this

8 Finding that balance is the real challenge for architects. It is quite difficult.

APPLICATION ARCHITECTURE ANALYSIS

ASSESSING EXISTING SYSTEMS

While many application components are bound early in modern development too, the effective use of constructs in modern programming languages creates the potential to push binding to a later stage. If the application architecture provides for sufficient decoupling and construction of concrete components through software factories, it is possible for the application to assemble itself at runtime, with different feature sets, without changing any source code⁹.

Ideally, the architecture is manifested in a framework that defines the coupling strategy and the framework provides for very late binding of components. If there is no framework on which the application is built, then the code should be analyzed for techniques that the programmer used to decouple the components from one another. Hopefully, the developer used familiar design patterns within the application. If not, there should be an expectation of difficulty in adapting to change and a corresponding increase in cost of making that change.

For example, I have developed a framework for security that allows authentication and authorization to be replaced within an application with zero code changes to the application itself. The security elements are defined when the framework starts up. The components for security can adapt to changing enterprise security requirements by implementing new techniques for security external to any application and changing the assembly descriptor for those applications to use the new implementation. Application security is considered to be a very deeply embedded part of an application design. If last-second binding can be done with security, it can be done with almost any definable part of an application.

Because a very small percentage of applications developers have strong skills in the use of design patterns for decoupling, it is important for enterprises to have a rich framework on which to build applications. This way, all programmers can leverage the skills of the senior developers and architects without having to be experts in those areas themselves. Still fewer programmers understand the power of late binding and know how to effectively implement those techniques in code.

If the applications being assessed are fortunate enough to have these strong underpinnings encapsulated in a framework, it is certainly a “keeper.”

⁹ That's a powerful statement. I would have never believed it was possible to do that 5 years ago, but I do it now on a daily basis.

ASSESSING SUPPORTABILITY

The most overlooked aspect to the ascertaining the total lifecycle cost of an application is the cost of supporting the code in a production environment. Without a doubt, modern applications are more complex than ever before. Most applications have a networked component to them. There is implicit networking involved with browser-based applications. Most applications that use a relational database or LDAP database access those services over a network, most likely to a different machine. The use of multiple-external networked services within an application means there are more components whose failure can have a direct impact on an application. When there are problems reported within an application, it can be a challenge for support technicians to determine if the problem is isolated to a single application or if a broader infrastructure resource such as a network segment or a database engine is the root cause.

Clustering adds an additional complexity. Clustering increases availability by introducing redundancy within the infrastructure. A single application may run anywhere within a cluster of multiple instances, most likely deployed on different machines. The value of increased availability for the applications comes at the cost of diagnostic complexity. Support technicians must determine if the problem is isolated to an application, an application within a single instance of a cluster, or all applications within a cluster.

The support costs also have a soft-dollar cost associated with availability. If the application isn't available to the user base, then there is a lost productivity cost. For direct retail applications, there can be an immediate revenue cost if the application is unavailable to take orders and execute the financial transactions.

In order to manage the cost of support, applications should be developed with robust diagnostic reporting capabilities. Logging facilities should report significant events within the application flow. The event information needs to be able to ascertain the machine, cluster instance and application under which it is running in order for diagnostic technician to gain a clear picture of the health of the application in the context of the infrastructure in which it runs.

As with many other architectural issues, the ideal is for the structural part of the application to provide this information so that applications developers can focus on business functionality. Again, the best implementation is for this structural

APPLICATION ARCHITECTURE ANALYSIS

ASSESSING EXISTING SYSTEMS

part of the application to be embodied in a framework. If that is not the case, the analysis should look for the consistency of operation event reporting and the quality of the information coming from those logging facilities within the application to determine the relative cost of supporting the application in a production environment.

CONCLUSION

Businesses are often faced with choosing between different software implementations. This may be a result of market or regulatory changes that radically change the business requirements for an existing application. It may be driven by the availability of new vendor software that can replace the maintenance of an existing application or it may be the result of a merger or acquisition.

Software clearly needs to be analyzed for the business functionality it delivers, but the analysis shouldn't stop here. The total lifecycle cost for software isn't just the development cost. The cost associated with changing requirements and operational support in a production environment contribute heavily toward whether software empowers a company or hinders it. The cost of support and resilience to change is directly related to the underlying architecture. While an architectural analysis is more challenging than a functional analysis, it is what will ultimately serve to reveal the optimal business decision as to which software base to support.

Author Information

*Gary Murphy
Hilbert Computing, Inc.
13632 S. Sycamore Dr.
Olathe, KS 66062*

*e-mail: glm@hilbertinc.com
voice: (913) 780-5051*