Turbo Charge Your Legacy Migration Project

Upfront Benefits Using an Incremental Process

Introduction

Many businesses know from experience that projects for:

- Legacy Application Enhancement
- Legacy Application Migration
- Legacy Application Integration with Web Services (Web Portal)

can be quite daunting and expensive. These projects require a huge investment in time, effort and money. The costs incurred to complete major work in a large core-business application – to make it deployable and functional in a production environment - can be quite substantial. Depending on the time it takes to finish developing, testing, and deploying the *entire* application, return on investment for the new application is often delayed for years.

For businesses that need to transition or enhance their monolithic legacy applications to a more agile environment where they can mix and match business logic into new applications, Crystal Point's transition-by-stages solution using AppViewXS 2.0 and the Struts Framework provides a nice answer. By overcoming the traditional inherent challenges of legacy application projects, businesses can decrease their cost before benefit gap, identify performance issues upfront, match development process to budget cycles, modernize and extend the life of their existing application.

The Challenges of Legacy Application Projects

Cost Before Benefit Gap Issues

Larger projects take more time to finish therefore the return on investment can be delayed significantly

The timeframe to implement projects for large, core-business legacy applications can be staggering; easily spanning years or even decades. The delay between incurred costs and achieved benefit is correspondingly large.

Minimizing that delay through incremental development has been difficult to achieve. When development spans entirely different operating environments – such as moving Guardian or Safeguard applications to Java - incremental development has been largely impractical. Therefore, no ROI occurs before the entire new application is fully developed, tested and implemented. The cost-benefit gap obviously, is huge.

Project Management Issues

Procuring money to completely finance large legacy application projects is extremely difficult

Businesses conducting major legacy application development projects face significant, costly, challenges that may result in going over budget and past deadlines. A major reason can be attributed to projects that are so huge in scope that they become too broad to manage efficiently.

If such projects slip past their phase deadlines, the anticipated future returns on investment decrease significantly. Unless real-world results can be achieved as the project progresses, any difficulty experienced in the overall project can place future funding in jeopardy, with the real risk of not receiving the necessary additional funding to complete the project.

Budget Issues

The costs involved for major work on core business Legacy Applications are very high. Since incremental development has been difficult to implement, this often means funding phases and milestones are equally difficult to define. This can result in larger funding increments, and increased difficulty in receiving incremental approval.

Legacy Application Maintenance Issues

Generating added value and benefit to existing legacy applications, or even conducting day-to-day maintenance of them, can be complex and difficult.

Finding programmers knowledgeable in some of the mainframe programming languages used to create legacy applications is getting harder (i.e., more expensive).

Often times, too, a legacy application has been so extensively modified over the years that additional change is very difficult to implement.

Performance Issues

The only performance that counts is the performance achieved in a production environment

Surprisingly, application performance may seem fine in a testing environment but, once in the production environment, perform poorly. The earlier such issues are detected and addressed, the easier they are to resolve. If such production-environment issues are not detected until rollout of the entire application, it can take an inordinate amount of time and resources to resolve performance issues.

No matter how development is structured and phased, unless ***production*** implementation takes place incrementally during development, large surprises tend to remain in hiding, usually in the areas of performance and modular interaction.

The Solution: Incremental Development via Legacy Application Web-Enablement and Transition Control

Incremental Development via Legacy Application Web-Enablement and Transition Control makes phased transition of legacy application projects possible

A solution is now available that addresses these issues. It is a Web Application that can communicate with existing Legacy applications, thus providing them web presence and modern GUI screens. This solution also permits rapid, flexible, and tightly integrated development of new applications across different environments such as Guardian and Java. The solution permits:

- Immediate and secure web-enabling of legacy applications
- Ability to automatically transform primitive legacy screens into modern web forms
- Ability to integrate legacy application into existing (secure) web infrastructure
- Concurrent use of both web-enabled Legacy application and developing Java application (or modules)
- Tight integration, i.e. seamless blending, between the legacy application and developing Java applications
- Efficient, Incremental transition from the web-enabled legacy application to the Java application

- An integration interface to support phased development and rollout of new applications or enhancements.

A multi stage transitioning approach makes legacy application projects much more manageable and feasible. Incremental Development starting with Legacy Application Web-Enablement and continuing as a phased, transitional process allows businesses to leverage their existing legacy application and focus on creating value added products.

This combined approach provides greater flexibility for businesses to meet their specific needs. Businesses using this approach can take on large projects ranging from Legacy Migration, Legacy Enhancement, and Web Portal Integration. Projects will be easier to implement and deploy over the various stages of the whole product development cycle.

Benefits to Incremental Development via Legacy Application Web-Enablement and Transition Control

Incremental Development via Legacy Application Web-Enablement and Transition Control reduces overall costs and makes unwieldy projects manageable.

Extend Legacy Application Life

If your legacy application's primary weakness is its primitive user interface, or its inability to be accessed over the web, both these failings can be addressed through a Web Application that provides secure, (platform-independent) thin-client browser access into your legacy application, and also for transformation of legacy screens into modern GUI screens. These capabilities alone can extend the useful life of your legacy application.

When it is realized that the web-enabled legacy application can be attached into existing web infrastructures, such as ldap security, the legacy application's continued value is further leveraged.

Finally, new Java development can be integrated seamlessly with the legacy application, so all add-on capability for the legacy application could be created in the Java environment.

Reduced Cost Before Benefit Gap results in quicker ROI.

Reduce Cost Before Benefit Gap

The approach of modernizing legacy applications in stages, by first web enabling them, followed by incremental transition to Java is one that greatly reduces the cost-benefit gap described earlier. First, secure web enabling provides immediate value. Automatic transformation of legacy screens into modern GUIs provides further immediate and dramatic value. Then, as Java development proceeds, transition from the Legacy environment to the new environment can be implemented in small increments.

By using a Web Application that provides a web interface into legacy applications and an interface to which other java applications can be attached, one gains the ability to transition gradually from the legacy application to the new Java application. It is possible to deploy a new, evolving application in an incremental fashion. No more waiting for the whole application to be finished before it can be used in a production environment. Thus, incremental development costs generate matching incremental results. The cost-benefit gap is closed. And, the incremental implementation of development phases means early phases can begin generating an ROI in the near term. It becomes entirely feasible that demonstrated results and ROI from early phases of the development could provide the funding for subsequent phases.

Manageable Projects

Large projects are hard to manage and difficult to keep on task. In reducing a large project into smaller definitive stages that have deliverables that can be used and built upon, the overall project will be easier to manage and track. And, as each staged deliverables is used in a production environment, it can provide important feed back to development managers to assist them in tracking and guiding the overall project effort.

Shifting budget cycles can have a huge impact on monolithic large scale projects

Fit into Budget Cycles

Large legacy application projects are hard to budget because they can last into years. By reducing large projects into smaller definitive stages (mini-projects), it is easier to map development stages to the budget cycle. This eases milestone delivery, obviously begins incremental funding approval, Also, if budgets get tight and development is placed on hold, any negative effect on the overall project status can be mitigated since the next phase of the project can be placed on hold until the funding becomes available while the previous project phase is already being used in the production environment, and thus achieving ROI.

Performance problems can be difficult to diagnose in monolithic application releases

Identify Performance Problems Up Front

Performance problems are hard to track down when testing a huge application that was developed without a staged transition. By using an incremental staged application release into the production environment, it is easier to pinpoint where the performance problem lies and they can be addressed immediately. Because each delivered stage must meet performance objectives, there will be no performance issues remaining when the times arrives for acceptance of the entire completed product.

Legacy Application Maintenance Issues

The approach of legacy application web-enablement, followed by gradual transition addresses most legacy application maintenance issues. First, web-enablement occurs as an exterior process from the legacy application – so no code changes are required in the legacy application. Second, since the transformation of Legacy screens into modern GUI screens is automatic, ongoing legacy maintenance can be unimpaired. Finally, since this approach permits integration of new java applications with the legacy application, all future large-scale maintenance (or development of add-on capabilities) can take place within the Java environment and either attach to, or replace legacy functionality.

Legacy Application Project Scenarios

There are many legacy applications projects that are currently being undertaken by various companies. Here are two examples that can benefit greatly from incremental stage development via Legacy Application Web-Enablement and Transition Control

Legacy Application Migration Scenario

At some point in time, businesses may determine that their legacy application needs to be re-written. An example of this is migrating an IBM legacy application over to the NonStop Java environment. Replacing an existing legacy application can require a huge investment in time, effort, and cost.

Using technology to provide immediate web-enablement and modern GUIs for the Legacy Application, with Transition Control, it is possible to deploy the new, evolving application in an incremental fashion. No more waiting for the whole application to be finished before it can be used in a production environment.

- First, provide immediate, secure web presence for the legacy application, utilizing modern GUI screens, and taking advantage of existing web infrastructure such as ldap access controls. .
- Then, begin 'mining' business logic from the legacy application and developing equivalent business functionality within a Java application.
- As elements of the Java application are developed, transition relevant transactions from the legacy environment into the Java environment, seamlessly blending the legacy application with the new application.
- Continue transitioning incrementally until the new application is completely phased in and the legacy application has been completely superceded.

In this fashion, Java application development becomes an interactive transition. The existing legacy application and the new re-written application will be running side by side using the Transition Control as the connection and control point to direct users into either the legacy application or the new application.

Legacy Application Enhancement and Integration Scenario

Providing value-added features within legacy applications can be complex and sometimes downright prohibitively difficult depending on how the application was designed, or how extensively it has been modified over the years.

By using Legacy Application Web-Enablement, a secure, modern web portal interface into your legacy application can be achieved in a *very* short time frame. The Transition Control aspect can also provide the integration point to incorporate 'external' java development as add-on functionality to legacy applications. This means one can modernize legacy applications without completely rewriting them. In addition, existing infrastructure and security modules can now be used to control access into the legacy application.

Crystal Point's AppViewXS 2.0 product integrated with Struts provides a powerful and effective tool for handling legacy application projects

The Crystal Point Solution: AppViewXS 2.0 and Struts

Crystal Point's AppViewXS 2.0 is a Web application that runs on the server side. It provides web access into legacy applications and has been integrated with the Struts framework to provide a convenient interface point to moderate the flow of information between the Web Browser and the Legacy Application. By using AppViewXS 2.0 with a phased transition project plan, legacy application projects are easier to manage and implement.

AppViewXS Background

AppViewXS is a server based product offered by Crystal Point to provide a method that is both powerful and simple to use to renew host applications and integrate for the web. This application provides tools for interface re-engineering of legacy applications. By "interface re-engineering" we mean:

```
Access to the functionality embedded within
terminal-based applications through graphical
user interfaces that utilize modern controls
and communication protocols to allow a
rejuvenated look and feel for the application
plus the ability to extend and build upon the
legacy application for workflow and interface
optimization as well as multi-hosted, multi-
application data integration.
```

AppViewXS basically acts as the conduit through which a "green screen" legacy system and its advantages can cross over to the GUI world of the Web. Without any programming or scripting, AppViewXS standardizes applications and makes them more intuitive and easier for users with varying skill levels to use.

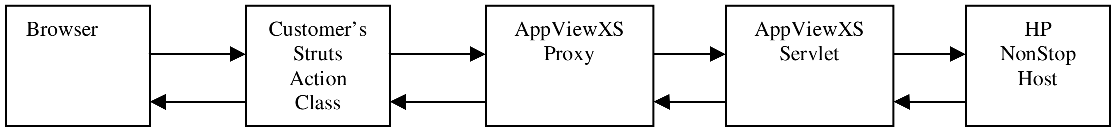
Struts Framework is a well known and established open-source framework used to integrate Java technologies and lets developers build web applications quickly and effectively

Using Struts to Provide the Plugin Point

Struts is an open source framework for building Java web applications. It is used to help extend and create a plug-in point to control access to the AppViewXS application which in turn provides access to the legacy application. This plug-in point provides a convenient and straightforward way to intercept the incoming data and process the result before sending it to the Host Mainframe or to the Browser. This is also the access point to leverage existing Security and infrastructure modules, and to enhance the synergy and interaction between a legacy application via AppViewXS and external web and java applications.

Struts Framework Process Flow Chart

To explain the process, a diagram depicting the flow of information through the Struts framework and AppViewXS 2.0 is shown below.



The Struts Action class provides the access point to external applications and other re-usable infrastructure related modules. The AppViewXS Proxy is responsible for maintaining the connection specific information as well as specific error handling to the AppViewXS servlet. The AppViewXS servlet provides the connection access to the HP NonStop Host and returns the HTML response back to the AppViewXS Proxy. AppViewXS Proxy parses the HTML response looking for specific connection information to update its state. The Action class then formats the response to parse the HTML data that should be displayed to the end user.

Benefits in using AppViewXS 2.0

Crystal Point's solution provides many benefits including very near time frame deliverable for legacy application projects, reduce cost before benefit gap, match development process to budget cycles, modernize and extend existing legacy application

AppViewXS 2.0 Web Application is already integrated with the Struts framework and it provides a useful plug-in point to combine the AppViewXS 2.0 application into your overall product solution. This solution provides many benefits and enhancements to any projects that require access to Host-based legacy applications.

With AppViewXS 2.0 and Struts, legacy application migration, integration, and enhancement can be achievable as a very near time frame deliverable. Coupled with incremental stage development cycles, large projects can be broken down into smaller projects and become easier to manage and monitor. The phased deliverables can be integrated and used in the production environment much faster and still leverage the existing legacy application.

While the benefits of the Struts framework integrated with AppViewXS 2.0 are great, it is also important to recognize and remember the other valuable capabilities AppViewXS continues to provide:

- AppViewXS 2.0 immediately provides a modern GUI look and feel to your legacy application (transparent to host application).
- Secure web-access to your host
- Load Balance connections evenly across different ports or hosts
- Dynamic Host Update

    - Provide a way to have AppViewXS session to display dynamic host updated messages onto a companion applet
- Tunneling Servlet
    - Provide a tunneling circuit to a host located in a different geographical location or network segment as that of the Web Application server.
    - Enhanced with a failover node in case the primary Tunneling Servlet goes down the secondary one will take over
- Create 'smart screens'
    - Increase usability via GUI controls
    - Add processing logic
    - Modify workflow
    - Data integration with multiple sources
- Built in Scripting Language – Advanced Macro Facility Script
    - Provide automation capabilities
    - Perform logical operations on data
    - Record/Playback Macro tool helps aid in generating custom AMF Script
- Make external calls to:
    - Java Applet
    - Class file
    - SQL to any ODBC/JDBC
    - HTML
    - XML
    - Open any URL – static or dynamic

Bottom Line

For businesses that need to transition or enhance their monolithic legacy applications to a more agile environment where they can mix and match business logic into new applications, Crystal Point's transition by stages solution using AppViewXS 2.0 and the Struts Framework provides a nice answer. By overcoming the traditional inherent challenges of legacy application projects, businesses can decrease their cost before benefit gap, identify performance issues upfront, match development process to budget cycles, modernize and extend the life of their existing application.