There are a number of choices open when looking at ways to take existing COBOL applications forward. In this document we will discuss the most common three – Extend, Replace, and Convert. In doing so, we examine the benefits and problems both real and perceived with each option and demonstrate the most cost effective method to modernize your application portfolio such that it can address your current and future business needs.
**Introduction**

The overriding pressure on IT managers is to do more in less time, spending less money, and to extract the most value from their business application portfolio. In taking those applications forward there are choices to be made between retaining and reusing the existing applications, replacing them with a packaged solution, or converting them to another language or platform from which greater value can be obtained. Each option has associated costs and risks that vary with the size of the application and the scope of the modernization ambitions for those applications. The issues and benefits remain fairly constant regardless of the size and complexity of the application so this document will focus on the issues and benefits rather than trying to derive cost estimates. That can be done in a more subjective manner once the issues and benefits have been identified. During the remainder of this document, the package solution can be regarded as any appropriate package, and language conversion is limited to conversion from COBOL to Java.

**Business issues with COBOL applications**

The COBOL language has been around since 1960 and has found its way into every type of business application. The main business issues that customers are concerned with along with the relevant responses can be summarized as:

**Can COBOL support the initiatives required to drive the business forward?**

The COBOL language has continued to evolve since its initial release. Today’s COBOL supports Object Orientation, XML data exchange, and integration with Java, .Net, and SOA infrastructures. It also runs on the latest platforms including 64-bit, and supports a range of contemporary databases. So any business initiatives that embrace the latest technology and platforms. As with other languages, developer productivity tools exist (“Integrated Development Environments” - IDEs) that ensure equivalent or greater programmer productivity on today’s COBOL platform as is experienced with other modern programming languages.

**Am I able to extract required information from the COBOL data that has accumulated over the years (data analysis)?**

COBOL has a number of proprietary data storage methods and solutions exist or programs can be written to provide analysis of that data. There are also a number of third party vendors that provide tools for creating customized reports without the need for further programming. In addition to this COBOL can be easily connected to a number of relational databases, using them to store and manipulate data. This means that the data analysis tooling provided by the database vendor can be used either directly or via ODBC/JDBC interfaces to extract the required information.

**Can COBOL applications provide the competitive edge required to support and expand the business?**

Because COBOL can integrate with many other languages and technologies, the required competitive edge can be created using COBOL or written in another language and then integrated into a COBOL application. This allows the creation of what are referred to as ‘composite applications’ where elements are selected because of the features and functionality they deliver and connected together to provide an integrated application.

**Will I have COBOL programmers in the future to continue supporting the core business applications?**

Just as we have ‘composite applications’, we’re seeing the emergence of ‘composite programmers’ i.e. programmers that are able to program in a number of languages who then combine the strengths of those languages to deliver feature rich applications that take advantage of the latest technology and platforms. As with other languages, developer productivity tools exist (“Integrated Development Environments” - IDEs) that ensure equivalent or greater programmer productivity on today’s COBOL platform as is experienced with other modern programming languages.

**Technical challenges with COBOL applications**

Although the COBOL language has evolved and can be used to create modern applications, there are some technical challenges that arise when dealing with applications that were written a long time ago, have been worked on by multiple programmers, and have been subject to numerous styles and standards over the years:

**The presentation logic (User Interface) and the business code are often intertwined.**

This makes modernization of the User Interface (UI) particularly difficult and often requires a re-organization stage to create the necessary separation of business logic and interface code before the UI can be modernized.

**A given application or program has evolved over time**

Depending on the discipline of the programmers or the standards imposed, applications can evolve in a non-standard manner making modernization more difficult because programs don’t conform to the same set of rules and need a lot of individual attention. Once this is discovered, projects and schedules need to be adjusted to cater for the additional time and work required.

**Application expertise has retired or been promoted out of the team**

Again, depending on the standards imposed, programs may not be well documented and supporting documentation may have been lost or misplaced over time. There are numerous tools available now that can scan and analyze application code and produce or recreate the missing supporting documentation.

**Application includes platform specific elements**

In the past; the idea of moving applications to different platforms was uncommon so often programmers would take advantage of functionality that was specific to the platform on which they were working. This may have been seen as a feature at one time,
but it presents an issue if that application needs to be moved to another platform. A decision has to be made to remove the specific function(s) or recreate the function on the new platform in a generic manner.

**Staff not motivated to work on older applications**

Programming environments have changed significantly over the years, particularly since the arrival of the PC. Integrated Development Environments (IDEs) have evolved and now we have products such as Visual Studio from Microsoft and the Eclipse IDE that support multiple languages and provide a common look and feel across them all. The addition of wizards and immediate syntax checking has made programmers more productive and allows them to produce more code of higher quality than ever before. They are motivated the functionality provided by the IDE and how it can assist them with their day-to-day tasks.

**Choices going forward (Extend, Replace, Convert)**

When attempting to decide a strategy for modernizing your existing application portfolio, there are a number of factors that need to be examined in order to determine if the benefits on completion of a given project outweigh the risks and costs associated with that project. These can be summarized as:

- Are there business issues that cannot be addressed using the current application language and/or platform?
- Can these issues be resolved by an off the shelf packaged product?
- Could these issues be resolved by converting to a new language and/or platform?
- Will you retain your current level of features, functionality and key differentiators encapsulated in the current applications?
- If not, is there a plan to remedy this once the package implementation or language conversion has been completed?
- How long will the package implementation or language conversion take and what will it cost?
- What happens to existing application during the transition i.e. how will requests for changes to the existing application be handled?
- What are the business advantages that will be realized on completion of the package implementation or language conversion?
- Will I have the same or improved performance (response times) once the package implementation or language conversion has been completed?
- If I currently have Service Level Agreements (SLAs) with my end user customers, will they continue to be met or improved once the planned changes have been made?
- Will I be able to compete more effectively with my competitors or be able to penetrate new markets that are currently unavailable to me?
- Will I realize improved bottom line benefits i.e. increased revenue, or reduced costs and what is the projected ROI?

**Extension considerations**

**Can COBOL support modern business application initiatives?**

As mentioned in the business issues section, the COBOL language has continued to evolve since its initial release. Today’s COBOL supports Object Orientation, XML data exchange, and integration with Java, .Net, and SOA infrastructures. It also runs on the latest platforms including 64-bit, and supports a range of contemporary databases. So business initiatives that embrace any of the above technologies or frameworks can be fully exploited by COBOL applications.

**Can all COBOL applications be extended?**

In general the answer is yes, but there are almost as many ways to extend applications as there are applications to be extended. The desired outcome often determines the scale of the extension project and whether the objectives will be achieved in a single step or as a sequence of individual projects.

**Are there risks associated with extending COBOL applications?**

Making any changes to an application carries some risk. They key is to ensure that the benefits gained from those changes outweigh any perceived risks so significantly, that the risks are minimized. By extending code that is already proven keeps the changes associated with the extension to a minimum which in itself reduces the risk significantly.

**New feature and function backlogs extend your time to market and can place the rapid enhancement capability required to improve competitive edge out of reach.**

**Package replacement considerations**

**Will the packaged application support all my business requirements?**

This is unlikely as packages are generic and will have the same features and functionality for everyone. It’s usual to have to spend some time adding the required functionality to the package in order to have a system that is comparable with the original application. This
extra implementation and integration work often extends the time and cost of a solution implementation and increases the Total Cost of Ownership (TCO) over the life of the replacement packaged solution well beyond the initial dates, budgets, and objectives.

**What support and maintenance complexity accompanies a replacement decision?**

Different languages, application integration components, time to market constraints (enhancement backlogs), all need to be considered when deciding on a replacement strategy.

**What business impact on my competitive time to market capability will a package have?**

New feature and function backlogs extend your time to market and can place the rapid enhancement capability required to improve competitive edge out of reach.

**How do I deliver new business initiatives during the package implementation?**

In an ideal world we’d be able to freeze all new requirements until the switch to the packaged solution has been completed, but given the extended period of time that these implementations take this is usually not possible. The often dynamic nature of the market place also means that we need to address new requirements during the implementation period. The result usually a cyclic extension of the overall implementation i.e. phase 1 is based on the system as it is at the start of the project and this is followed by additional phases to pickup and implement and required changes that were required during the prior implementation stages. This does of course extend the time and cost of the full implementation and is often a contributing factor to the eventual failure of long running implementation projects.

> The key to understanding this code is the process of excavating the architecture of the Java system.

<table>
<thead>
<tr>
<th>Original Application</th>
<th>Transition</th>
<th>Resulting Position</th>
</tr>
</thead>
</table>
| Extend               | COBOL applications with unique selling point and business value built in | Micro Focus Development tools, Visual Studio, Eclipse, other languages such as Java, C# etc | Re-use of original code  
Easy to maintain  
Full control over future enhancements and changes  
Existing programming skills can still be used |
| Package              | COBOL applications with unique selling point and business value built in | Configure and set-up environment for new package  
Recreate unique selling points and business functions not present in package  
Customize package to suit end user requirements | Completely new application with customized elements  
No direct control over future enhancements and changes  
May need new skills to maintain new application |
| Convert/Translate    | COBOL applications with unique selling point and business value built in | Automatic ‘black-box’ converter, usually via iterative process and possibly with manual intervention | Completely new application in new language  
Generated code is not easy to maintain and not conformant to OO language standards  
Full control over future enhancements and changes providing the generated code can be understood  
Existing skills cannot be used with new application |

Figure 1 – Summary of options and related issues
Conversion considerations

Will this conversion effort deliver anything more than I currently have?

“Functional equivalence” is usually the initial conversion target and can be highly automated resulting in a modern application with a demonstrably lower cost of ownership. The conversion effort itself creates the competencies required to maintain the converted applications and establishes an accelerated time to market capability.

Is Java just another mature language?

“Java, the oldest ‘new’ programming language around, is falling out of favor with developers. When it comes to developing the increasingly common rich Internet applications, Java is losing ground to Ruby on Rails, PHP, AJAX and other cool new languages. And there are even reports that Microsoft’s .Net, is pushing Java out of the enterprise” (source: Bill Snyder, Infoworld 28 December 2007). So the question that springs to mind is ‘Why would someone be porting an application from one “mature” language to another?’ Remember, Java is the COBOL of the 21st century! This question is revisited in the summary at the end of this document.

What are the issues involved with converting COBOL to Java?

Although we’ve had Object Oriented (OO) COBOL for some time, the majority of COBOL applications are procedural and do not make use of OO features. Java on the other hand is entirely object oriented so there are issues to be resolved in converting from a procedural to an OO paradigm. The application code needs to be analyzed in order to identify and resolve these issues. The analysis can be done either on the original COBOL code, or on the converted/translated Java code and follows common steps regardless of which code set is analyzed. In the example here, the analysis is applied to the converted Java code.

Identification

During this phase the domain model of an application is recovered and captured. The domain model consists of business entities and business processes. This domain model will determine how the Java code needs to be re-factored to create a more robust architecture, in which the proper business entities and their corresponding operations are adequately represented by Java classes. Once the domain model is identified, the horizontal, system services are defined: technical infrastructure (middleware services, or security, etc.), and small-grained, reusable business services (currency and text management services, for example).

Understanding

Code level quality assessment. This step will provide an understanding of the Java generated code and evaluate the quality of the Java code and its object-oriented structure.

Design level assessment. The next step will detect anomalies in the generated Java code that decrease its maintainability and architecture robustness.

Architecture level assessment. The next step is to understand the baseline architecture of the generated Java code. The key to understanding this code is the process of excavating the architecture of the Java system. The excavation methodology is a systematic process of creating a high-level logic model that reorganizes the Java packages and classes into architecture layers according to the selected target architecture. The process of excavating the architecture of the Java generated code will use as the starting point the identified business entities, workflows and common system services. The target of a high level architecture is that there should be 4 main architecture layers: 1) Presentation, 2) Business Logic, 3) Data Integration, 4) System Services. These four layers will facilitate the transition towards the proper J2EE architecture.

Domain level assessment. The fourth step is to correlate the identified business entities and processes to the classes within each architecture layer. The classes that are identified for each entity or process are grouped together into a conceptual component. The conceptual interface of the component can then be defined and published. This is the interface that corresponds to the business operations.

After the above assessment steps have been performed, the roadmap for improving the Java code should be developed. The roadmap is determined by integrating suggestions from each step within the context of business objectives of the project. A set of integrated rules is applied to the output of the above four steps. In order to check conformance to the integrated rules all the above deliverables are required at the same time: the conceptual components, their conceptual interfaces and high level architecture as well as the design level and code level anomalies. Both the integrated rules and the above assessment steps are part of the overall methodology.

There are four categories of integrated rules:

- Each business entity should correspond to exactly one conceptual component in the Java code. For instance, if one conceptual component relates to two business entities, this component needs to be refactored.
- The interface of each component should be well organized. For instance, a data member should not be accessed directly but through setter and getter methods.
- Functionality should be well organized into architecture layers. For instance one class should not contain a mix of presentation and business logic. Between the architecture layers, the classes in the business logic layer should never reference the presentation layer.
- The J2EE platform should be effectively utilized. This will involve analyzing the generated EJBs to ensure they conform to standard EJB coding standards. In addition, identifying any generated system services that could be replaced by services provided by the Java platform, for example, data types, authentication or logging should be utilized rather than separately developed services brought forward from the original COBOL code.
Conclusion

Modernization of mature applications can be a very complex, costly, and risky undertaking. From the outset the goal should be to minimize these three elements as much as possible to ensure a successful outcome. The greater the scope of any planned changes, the greater the impact on these three elements. If you only change 10% of your application code, the risk is contained within that 10%, or put another way, there is 90% of your code that is subject to no risk at all. Any solution that results in a total change of your application magnifies the risks, costs, and time allocated to completing the project exponentially. This is particularly true when code conversion/translation is required and worse still when the two languages involved are very different in their core structure e.g. procedural vs. OO. If the current application has a dated look and feel, then the converted application will also have a dated look and feel and will be in a mangled and fairly unmaintainable state. So the start point from which the look and feel of the application can be improved is now far worse than the previous position. Reusing the existing application is a more viable start point because less of it has been interfered with and it will be easier to maintain because it has not been pushed through a rules based code converting black box, nor subjected to the ‘many hands make light work’ approach of offshore outsourcing.

A packaged solution starts off as the same application for all its users, so you have to invest significant amounts of time and money to get to the same level of functionality as the application its replacing. Giga reports that 70 percent of Package Replacement projects fail.1 The Standish Group reports 72 percent of rewritten business applications fail or are seriously “challenged.”2 Given COBOL’s ability to integrate with technologies such as SOA, .NET etc, existing applications can be modernized by integrating specific components with AJAX, C#, Ruby on rails, VB.Net etc. The high risk ‘all or nothing’ replacement and conversion approaches can be sidelined in favor of more selective less invasive modernization techniques.

Written by Irving Abraham, Field Solutions Director, Micro Focus.

About Micro Focus

Micro Focus, a member of the FTSE 250, provides innovative software that allows companies to dramatically improve the business value of their enterprise applications. Micro Focus Enterprise Application Modernization and Management software enables customers’ business applications to respond rapidly to market changes and embrace modern architectures with reduced cost and risk.