



## Re-architecting Legacy Applications

by

Andrew McAllister, Ph.D.

Vice President – Client Services, Trinity Millennium Group

Organizations considering modernizing their application portfolio often face a seemingly insurmountable dilemma. On one hand, a timely and cost-effective migration path is critical to support a positive return on investment and meet significant milestone dates, such as impending license renewals or critical resource retirements. This seems at odds, however, with the need to re-architect the applications. Older systems are often written in procedural languages with text-based user interfaces and non-relational databases. Years of maintenance patches commonly leave the source code in a brittle, poorly organized, and sparsely documented state. Re-organizing such complex applications to be consistent with modern industry standards can be a monumental task, which seems at odds with the need for a fast solution.

When completion time is an overriding concern, some organizations turn to off-the-shelf code conversion tools. Such tools can produce compilable, functionally-equivalent versions of the applications for modern programming platforms such as .NET and J2EE. The problem is that off-the-shelf tools tend to perform simplistic line-by-line translations, where the new programs mimic the structure of the legacy applications. The new code looks generated and does not conform to current programming standards. Programmers familiar with established object-oriented best practices find such code difficult to understand and maintain, which increases long-term maintenance costs and decreases job satisfaction.

This suspect level of quality is one factor that leads some organizations to turn to a more traditional solution, manually re-developing the applications. This option offers the allure of a familiar process and full flexibility for re-architecting. Unfortunately such projects tend to be lengthy and expensive. Three main factors contribute to the large effort required:

1. Typical business applications involve millions of lines of source code with many complex inter-dependencies. Manually sifting through the myriad details is an immense task;
2. A legacy application typically undergoes many years of enhancements, which add considerable functionality. A re-development project will necessarily be larger than the original development project;
3. For a green-field development project, the new application must satisfy the documented requirements. The measure of success is more demanding, however, for a modernization project. The new application must account for every bit of detailed functionality in the legacy application. Verifying that this is the case (and applying fixes when it is not) consumes considerable time and effort in the latter stages of manual re-development projects, and is a common reason for budget over-runs.

This paper describes a newer approach called **Automation-Enabled Modernization™** (AEM), which combines the speed and cost-effectiveness of automation with the flexibility for re-architecting according to the unique needs of your organization.

## **The Need for Flexibility**

Experience has shown that re-architecting requires application-specific knowledge about the legacy software. To illustrate this, consider as an example the conversion of security functionality found within older applications implemented in a technology such as Natural / ADABAS. One could examine several such applications and find that each one has a different way of determining which users are permitted to perform particular actions or access specific screens. One application might accomplish this using "IF" statements scattered throughout the legacy source code, while a second may use a table-driven approach where permissions are extracted from the database. Still another application might have all security functionality centralized within a single module that is automatically called as part of each attempted transaction. Each of these applications uses exactly the same technology, yet they employ completely different approaches to accomplish the same type of functionality.

Planning a conversion typically includes defining a specific approach for implementing security as part of the desired new application architecture. Perhaps a third-party security package will be used, so the legacy security functionality must be converted to data records in the format required by the package. Alternatively, the target architecture might call for security conditions to be implemented as VB.NET code within an isolated security layer.

One ramification of these observations is that re-architecting requires flexibility in:

- a. recognizing and extracting patterns that occur within a given legacy application; and
- b. mapping to new types of artifacts that are specific to a given target architecture.

An off-the-shelf code conversion tool simply cannot provide this level of flexibility. By necessity, such tools are designed solely based on knowledge of the source and target programming languages. Such a tool can recognize an "IF" statement in one language and convert to the equivalent "IF" statement in the target language, but the tool has no chance of recognizing types of functionality that are independent of the language features used to implement them.

## **Automation-Enabled Modernization™**

Automation-Enabled Modernization™, or AEM for short, provides the required flexibility by employing a highly configurable tool set. This approach allows a unique set of legacy-to-new mappings to be defined and implemented for each transformation project.

The process begins with automated source code parsing, which populates a repository of metadata to be used during transformation. This automated processing is labeled as Transformation Prerequisites in Figure 1, and creates artifacts such as abstract syntax trees and symbol tables. All information present in the source code is captured during this step.

Next a combination of automated searching and manual code sampling is used to identify patterns within the code associated with various types of functionality. Such a pattern is said to represent a "slice" of the source code. For example, analysis might reveal a large number of "IF" statements that include the phrase "permission\_level." The collection of all such statements found in the source code represents one slice, which in this case is associated with the security aspect of this application. This type of analysis determines not only how security is implemented, but also data access, user interfaces, business processing logic, and other aspects of the legacy application. Subject matter experts familiar with the legacy code are often helpful in identifying code patterns of interest.

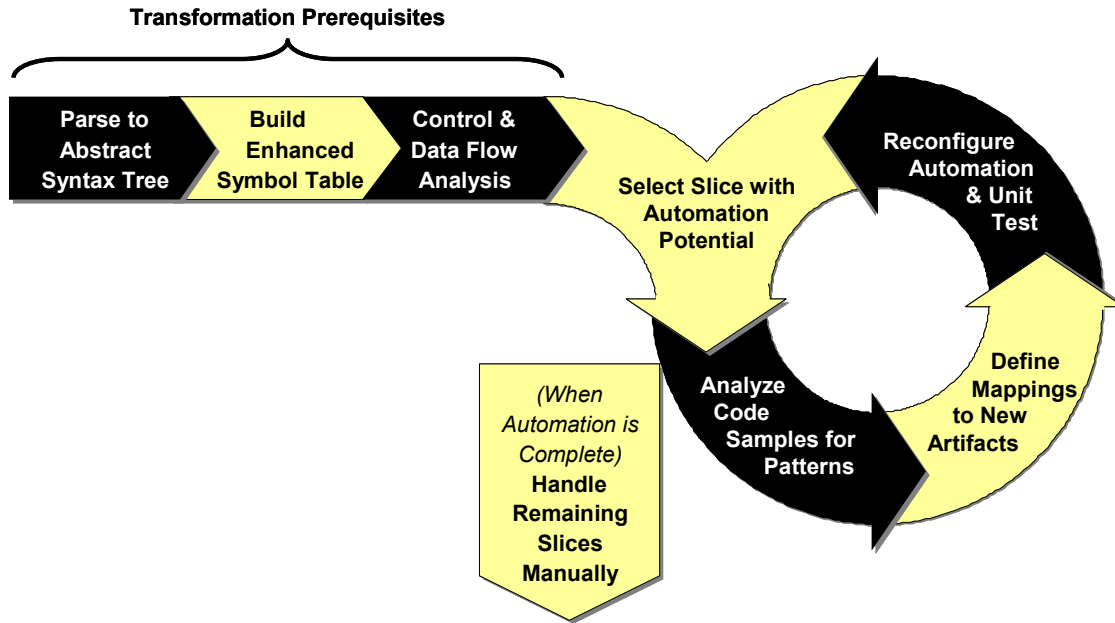


Figure 1: Applying Automation to Application-Specific Coding Patterns

A further step is to decide how each slice should be mapped to an appropriate implementation in the new application. Depending on the target architecture, the "permission\_level" conditions described above might be mapped to equivalent "IF" statements, or perhaps to data records representing the user permissions.

As Figure 1 shows, the automated tool set is then configured to transform this particular slice. Tools must be able to:

- recognize patterns within the legacy source code (or more accurately, within the metadata repository);
- transform code instances to functionally-equivalent new artifacts;
- insert the new artifacts into the appropriate portions of the to-be application architecture; and
- keep statistics and traceability data as a record of which code has been transformed so far.

Traceability data is critical to this approach. There is a need to be able to tell with certainty what has become of every line of legacy source code. Similarly, one must also be able to identify the derivation of every line of newly created source code. Such data serves two important purposes. This is the only way the modernization team can track progress and confirm that the transformation is appropriate. Equally importantly, the client receives proof that (a) all legacy functionality has been transformed, and (b) no unnecessary new functionality has been introduced.

This type of automation provides dramatic reductions in programming effort. It is common to implement a slice, run the automated transformation, and find that tens of thousands of instances have been transformed. Eventually the return on investment from automating the slices tends to lessen. For a slice with only a handful of instances in the source code, manual conversion of the code may be a more cost-effective approach. Moreover, some source code is

problematic and may be more effectively converted by hand. An example might be a complex snarl of unstructured GOTO statements. As indicated in Figure 1, at some point in each transformation project a decision is made to stop automating slices and convert the remaining code by hand.

Experience shows that 80 to 90 percent of the code can typically be transformed with automation. For applications containing millions of lines of source code, this results in dramatic reductions in effort, time, and cost when compared with manual re-development. Furthermore, since the newly transformed artifacts are created one slice at a time, they can be used to populate the appropriate components of the target application architecture. Monolithic programs produced by off-the-shelf conversion tools cannot support re-architecting in this manner.

Figure 2 summarizes several comparison points for the most common application modernization approaches. In general, manual re-development enables ultimate flexibility but comes with a high price tag. This approach fails to take advantage of the years-long investment in creating the legacy functionality. Off-the-shelf tools can produce code quickly, but without re-architecting to the modern standard of quality required for effective maintenance. AEM offers the best of both worlds – automation to achieve an attractive return on investment, together with flexible re-architecting.

Item	Common Requirements	Application Modernization Approaches		
		Manual Re-write	Off-the-shelf Code Conversion Tools	Automation-Enabled Modernization
1	Take advantage of unique legacy code characteristics	✓	✗	✓
2	Flexibility in mapping to a client-specific architecture	✓	✗	✓
3	Ease of maintenance for long-term ROI	✓	✗	✓
4	Leverage automation for effort, time, and cost reduction	✗	✓	✓
5	Handle several legacy languages	✓	✗	✓
6	Complete traceability between old and new source code	✗	✗	✓

 Meets the requirement  Typically does not meet the requirement
---

Figure 2: Application Modernization Approach Comparison Matrix

**About the Author**

Andrew McAllister, Ph.D. is Vice President of Trinity Millennium Group, an international provider of automation-enabled application modernization services. He has 30 years of IT experience as a consultant, researcher, professor, and executive. He is an acknowledged expert in automated software engineering technologies and has published dozens of related articles.

Email: [ajmcallister@tringroup.com](mailto:ajmcallister@tringroup.com)

Website: [www.tringroup.com](http://www.tringroup.com)