# Guide to Successful Software Modernization

**Great Migrations**
SOFTWARE TRANSLATION TECHNOLOGIES

April 2015

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to set the context for software modernization work and to help outline the information needed to plan and estimate a successful software modernization project using Great Migrations (GM) tools and methods.

## 1.2 Audience

This document will be most helpful to individuals who are planning a software modernization project. This document assumes the audience is familiar with the Great Migrations methodology and related concepts described in The Great Migrations portal. This document will be complemented by deliverables produced in a Smart Start engagement with Great Migrations.

# 2 Modernization Project Objectives

## 2.1 Adapting to Changing Technologies

The purpose of a modernization project is to upgrade legacy applications to use new technology. An example of a modernization project would be to begin with an application written in Visual Basic 6 using various COM libraries (VB6/COM) and upgrade it to a functionally equivalent, or functionally superior, application written in C# using the .NET framework (C#/.NET). In this example, we would be completely changing three fundamental foundation technologies:

- The programming languages used to code system, and
- The various runtime libraries and frameworks used to access external services
- The tools used in developing with the application

The programming language dictates how developers can describe data structures, interfaces, and algorithms. The libraries provide an extensive array of advanced services to the program such as data access, communications, and graphical user interface. Tools are the software that developers use to help them organize, read, and modify the system code and to debug, build, and package it for use by the business. Together, the language, libraries, and tools create the "development platform". The platform sets the rules and makes the system possible.

Languages, libraries, and tools inevitably evolve and are replaced by next generation technologies. Usually, such changes are localized and gradual with an appropriate measure of backward compatibility so developers can adapt the system through standard maintenance activities. However, sometimes, the platform changes are broad, radical and disruptive. Such is the case with Microsoft ending support for VB6 and upgrading Windows development to .NET.

## 2.2 What is your Business Case for Modernization?

Failing to adapt to changing technology leaves your systems on an unsupported platform which leads to business problems such as the following:

- Staffing challenges: problems finding and retaining qualified technical staff
- Integration challenges: problems integrating your systems with external technologies
- Consolidation challenges: problems unifying and standardizing technical assets and processes
- Inability to adapt: problems adapting to changing business and technical requirements
- Inability to innovate: problems modifying systems in creative ways
- Increased cost and risk: problems controlling costs of IT and business operation
- Eventual failure: business outages due to technical incompatibilities and staffing challenges

An effective modernization effort moves your software assets to a supported platform and addresses each of the problems above to produce several benefits:

1. The business functions and features of the legacy systems will be accurately preserved or improved and be maintainable on a supported platform.
2. The development team can continue with maintenance work during and after the upgrade. The team is also able to take advantage of the new tools and capabilities of the new platform. This enables a boost in agility and productivity.
3. The organization's IT landscape is attractive to a larger pool of developers, technology vendors, and related products and technical knowledge. This improves their ability to recruit and retain qualified technical staff.

Modernization benefits are difficult to quantify and their financial impacts will vary across organizations. In many situations, the business case is made on qualitative grounds: The organization has consensus that running their systems on an unsupported platform creates a growing risk to their business. And, they recognize that modernizing the systems, the people who maintain the systems, and the processes used to maintain the systems is a highly effective and efficient means of mitigating that risk while serving a broader application lifecycle management strategy.

## *2.3 What makes a Modernization Project Successful?*

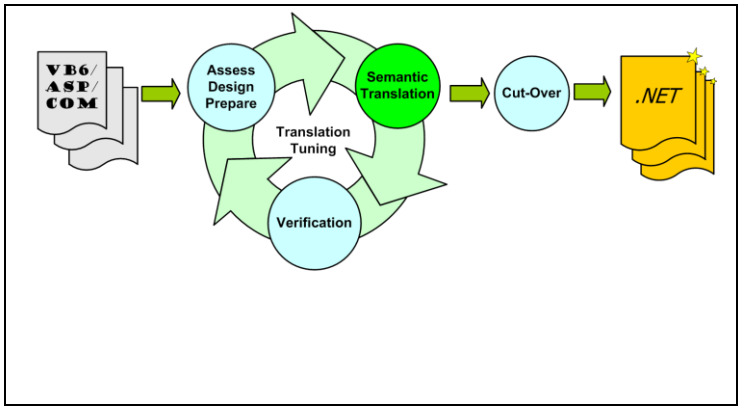
The key success factors of a modernization project are listed below:

1. Accurately preserves legacy functionality and features, unless explicitly identified for change or removal
2. Makes technical and architectural improvements
3. Produces a new system that is as maintainable or more maintainable than the legacy system
4. Causes minimal disruption to the users of the system and to other ongoing maintenance and project work
5. Meets or exceeds the quantitative business case if one is available

# 3 Project Methodology Overview

Moving software from one technology platform to another is potentially the most radical and disruptive type of maintenance an organization can undertake. In order to meet the challenges it presents, a more strategic and extensive approach is needed. Great Migrations calls our approach to platform change the **Tool-Assisted Rewrite Methodology**.

The defining feature of this methodology is a staged, iterative approach and its use of a sophisticated modernization tool, **gmStudio**, to help read, interpret, and rewrite the legacy source code according to the project team's unique requirements.




Used properly, the Tool-Assisted Rewrite methodology produces a repeatable upgrade solution for your systems that integrates tool-generated code, hand-written code, and third-party components. The methodology balances automated and manual work to improve accuracy, efficiency and scalability while allowing high-visibility of progress and full control over results.

A detailed discussion of the inputs, phases, tasks and deliverables of the Tool-Assisted Rewrite is beyond the scope of this document, but additional details may be found in The Great Migrations portal.

**Convert or Rewrite?**

"Convert or Rewrite?" is an important question that comes up every time an organization is planning a modernization effort. Unfortunately, these two options are poorly understood. The fact is the VB6/COM platform is dramatically different from .NET; and moving functionality from VB6/COM to .NET presents many needs and opportunities for technical rework. So, the question is not "convert or rewrite?" it is "how should we rewrite?" and how do we ensure success.

While some parts of a legacy system might need a completely fresh start, this will typically be the exception. Most mature applications are delivering business value, many are considered critical to the business, and their functionality must be preserved. This is particularly important in a platform upgrade because so much is changing for technical reasons. That is why we believe successful teams should leverage tools to help them analyze and interpret their legacy codes and systematically and precisely restructure that code while accurately re-implementing it for the target platform.

Furthermore, because of its flexibility, using gmStudio does not constrain the final results. However, other factors: limited time, limited budget, and lack of clarity of technical vision -- will constrain your scope. Fortunately, gmStudio allows the team to work faster, experiment with options, and scale up. Teams using gmStudio effectively will be able to do more re-engineering, do it much faster, and do it much more accurately, within a given budget and schedule. Ultimately this translates into better results and greater value. A "major rewrite" does not need to be a "from scratch" rewrite: the tool-assisted rewrite with gmStudio can help you produce the same results with much less labor, disruption, and risk.

# 4 Modernization Planning Guide

This guide organizes modernization planning information into four areas:

1. **Project Information**: top-level information about the project and key milestones
2. **Functional Requirements**: an outline of the finite set of testable functions desired in the new system
3. **Technical Requirements**: descriptions of technical upgrade features expected in the new system
4. **Organization Plan**: descriptions of various work management related matters

This section is only an abstract guide to the types of information to gather and the questions to ask and answer. The information used to execute actual migration work will be concrete and detailed and will be organized into separate design and strategy documents, spreadsheets, and/or a formal requirements/collaboration system.

*In the following sections, specific information gathering tasks and subtasks are bullets and sub-bullets. For example:*

- *Information gathering task*
  - *Information gathering sub-task*

## *4.1 Project Information*

- Complete the following table of top-level project information.

| | |
|---|---|
| Name of Project | |
| Desired Start of project | |
| Desired Duration of project | |
| Project Stakeholders | |

### 4.1.1 Project Phases and Milestones

The critical milestones of a tool assisted rewrite methodology correspond to periodic, typically monthly, delivery of the new system produced by the upgrade solution and measured according to specific quality metrics. Stakeholders may want to define additional milestone targets and deliverables for the effort.

- List any specific milestones you require for the project plan.

| **Table: Milestone Chart** | | |
|---|---|---|
| Milestone Name | Target Time | Description / Deliverable |
| | | |

## *4.2 Functional Requirements*

The fundamental functional requirement for a modernization project is this:

*The new system must be functionally equivalent, or superior, to the legacy system.*

In order for this vast, vague requirement to become useful for work management, it must be broken into a finite set of testable functions and organized into an outline.

- Create a hierarchical outline of the testable functional requirements for the project. For each requirement supply the following attributes:

  Name — The names of the requirements in the following hierarchy: application, major functional area, minor functional area, testable function, alternative scenarios.

  Objective — The new system may omit legacy functionality that will not be carried forward; it may have functionality that is expected to change, and it may add functionality that will be new. Indicate this modernization objective for each function as follows:
  - Match: Indicates a function should match the corresponding function in the legacy system,
  - Remove: Indicates the function was in the legacy system but will be omitted from the new system
  - Change: Indicates a legacy function should be changed
  - New: Indicates a function should be added to the new system

Testability Each function must be testable. The test readiness for each function may be graded as follows:
- Undocumented
- Documented with execution steps specified
- Documented with execution steps and input data specified
- Documented with execution steps, input data detailed, and verification steps/results specified
- Fully Documented and Automated

## *4.3 Technical Requirements*

The primary focus of platform modernization project is the technical upgrade. Like the functional requirements, the technical requirements can be organized into a hierarchy of measurable upgrade and redesign features. Technical requirements are organized into several areas:

1. Legacy Code Metrics – Source Structure Model
2. Legacy Platform Dependencies and Upgrade Strategies
3. Optional Technical Requirements

A framework for technical requirements may be created by an automated process using gmStudio to inspect, interpret, and report on the legacy system. This information may be loaded into a SQL database for additional consolidation, transformation, and analysis. This framework will contain a vast body of detailed information; however, it is only a framework. Additional analysis and creative thinking is needed to add critical information such as: purposes, upgrade strategies, and other insights. Useful technical requirements require subjective input from people who are familiar with the legacy code, the legacy application roadmap, and the organization's rules, standards, and guidelines for building new software on the target platform.

### 4.3.1 Legacy Code Metrics

One of the key artifacts used in planning a legacy modernization effort is the detailed **As-Is Inventory** of the legacy system. These metrics include data about the structure and size of the legacy system. This information produces a **Source Structure Model** that will help organize other information, and activities like redesign and verification.

- Tabulate the name, order the type each VB6 Project (VBP) that is in scope
  - Types: OleDLL, OleEXE, Control, EXE
  - Organization: desired order and grouping of applications and related VBPs
  - Identify any cycle in the build order of the VBPs
- Tabulate Numbers, Types, and Composition of Files (source structure model)
  - Types: Form, Module, Class, Control, Designer, Property Page, Binary (FRX, CTX, RES)
  - Composition: Blanks, Comments, Logic, GUI
- List Dead Code
  - Identify unused methods that may be removed
- List Cloned Files
  - Identify Copies of very similar or identical files that may be consolidated
- List Shared Files
  - Identify Multiple references to the same files that may be centralized on a common framework
- List Duplicate Code
  - Identify Copies of very similar or identical methods that may be consolidated and refactored
- List Numbers, types, and size of Business Logic Methods
  - Specify Functions, Subs, Properties, Declares, Event Handlers and usage data for each
- UI Complexity
  - Specify Numbers and types of controls on each form or control
- Functional Mapping (manual input required)
  - Relate the source files and contents to the items on the functional requirements outline.

### 4.3.2 Legacy Platform Dependencies and Upgrade Strategies

#### 4.3.2.1 External COM Components

One of the most important drivers of effort in VB6/COM modernization is upgrading COM components and controls to appropriate .NET replacements. There are three parts to doing these replacements:

1) Gathering details regarding the COM dependencies (including dependencies on .NET through interop)
2) Defining COM upgrade strategy that describes how COM services will be provided by .NET

3) Implementing and verifying the upgrade strategies; in many cases this can be automated by creating various gmStudio rules files as described in this article on COM replacement using gmStudio.

- Gather metrics for the COM Libraries, Controls, Designers defined in third-parties components
- For each reference to an external COM symbol, list the full identifier, member type, usage patterns, and location
  - Summarize usage counts by API member
  - Summarize the purpose of each class
  - List desired replacement at the class/member level and describe your past experience and standards for the replacement if any. This is an extremely critical aspect of upgrade planning.
  - Identify Special Licensing / Installation requirements
  - Identify Special purpose? (i.e. VB6 Plug-ins, sub-classing APIs, etc.)
  - Identify Interop Components (i.e. references to the .NET framework through interop)

#### 4.3.2.2 Internal COM/COM+ Dependencies

Another important driver of effort in VB6/COM modernization is the intra-dependencies among parts of the applications. This information impacts how the work is ordered and also identifies opportunities to improve how different parts of the legacy application interact with one another.

- Gather metrics for the COM Libraries, Controls, and Designers defined in your VB6 source code
- For each reference to an internal symbol, list the full identifier, member type, usage patterns, and location.
  - Summarize usage counts by API member
  - List desired replacement at the class/member level.
    NOTE: Usually internal classes use the "identity" replacement strategy: the upgraded class becomes its own replacement; however in some cases, an internal class can be replaced more effectively by pre-built code or a binary component from a third party or from another .NET development effort done within your organization.

#### 4.3.2.3 Win32

Entry-point (a.k.a. Win32 SDK) API dependencies are another legacy platform dependency that most teams would like to avoid when upgrading to .NET.

Entry Point-API dependencies vary in if/how they can be upgraded when moving to .NET. In some cases, .NET framework classes are available to provide the legacy SDK services in a managed way; this typically requires some manual rework. In other cases, the new code will continue calling legacy APIs but some adjustments may be needed to ensure those calls work correctly. It is also common for a code to declare APIs and supporting data structures that are not used. If desired, these unused elements may be identified and removed with the help of gmStudio as part of the Dead Code cleanup.

- Gather metrics for the entry point APIs used by your source code
  - For each reference to an entry-point API list the full declaration, purpose, usage count, and usage patterns.
  - List desired replacement if available and describe experience and standards for using the specific replacement technology if any.
  - List entry point API declarations and supporting structures that may be removed.

#### 4.3.2.4 Language Incompatibilities

The VB6 language and runtime environment are very different from .NET languages and runtime environment. Most VB6 applications have features that can only be preserved either by redesigning and reworking the application code or by using a VB6 emulation framework to provide backward compatibility. Gathering information on the frequency and distribution of incompatibility issues in the legacy system and formulating upgrade strategies that meet the needs and standards of the organization is an important task in planning the upgrade effort.

- List the distribution and usage of various VB6 language features that may attention in .NET such as:

| | | |
|---|---|---|
| ▪ App Object<br>▪ As New<br>▪ Collections<br>▪ Conditional Compilation (#if, #else, #const)<br>▪ Deterministic Finalization<br>▪ Data Binding<br>▪ Data Designer | ▪ Drag and Drop<br>▪ Dynamic Code (TypeName, TypeOf, CallByName)<br>▪ Error Handling<br>▪ File IO: text, binary Gosub/Goto<br>▪ Graphics Functions (on Form, on Printer)<br>▪ Implements<br>▪ Implicit type conversions | ▪ IsMissing<br>▪ Null handling<br>▪ Printing and Printer Object<br>▪ Property Pages<br>▪ UserControls<br>▪ UserDocuments<br>▪ Weak Types (Variant, Object, Form, Control, Any) |

#### 4.3.2.5 Configuration Management Requirements

Configuration management relates to version control, builds, and deployments of a system. A large platform upgrade introduces a vast amount of new code to be versioned, as well as new tools and standards to be used for builds and deployments. Consequently, configuration management changes must be defined and planned as part of the project.

- Describe version control model for legacy applications
- Describe desired version control standards for new code
- Describe standard build process for legacy applications
- Describe desired standard build process for upgraded applications
- Describe packaging, installation/uninstallation requirements for legacy applications
- Describe desired packaging, installation/uninstallation requirements for upgraded applications

#### 4.3.2.6 Performance and Scalability

Moving an application to a new platform may introduce unexpected changes in performance and scalability. Consequently, performance and scalability requirements must be defined and planned as part of the project.

- List and describe specific performance objectives and measures for the new application. These can be documented as an additional piece of information on the functional requirements outline.
- Describe your current stress/performance testing processes and capabilities
- Describe hardware, network and infrastructure considerations that will impact stress testing

### 4.3.3 Optional Technical Requirements

In addition to the core technical upgrade requirements in a given VB6 to .NET upgrade, there are many potential improvements that organizations may want to include in their project. Including these requirements goes beyond the must-have requirement of making the application build and run correctly on .NET. These requirements move into more subjective value-added, should-have and nice-to-have design features and process improvements. Adding these features to your project may have a big impact on the effort and risk of the project as well as its potential benefits. Some examples of optional requirements are listed below.

#### 4.3.3.1 Application Architecture Changes

- Elaborate application architecture improvements that are in scope for this project.
  Possible examples are the following:
    - Redesign Business Object Model
    - Remove undesired coupling across architecture layers
    - Remove maintenance and support “Hot Spots”
    - Improve testability of areas
    - Implement a new strategy for application security

#### 4.3.3.2 System Architecture Changes

- Elaborate on system architecture improvements that are in scope for this project.
  Possible examples are the following:
    - Upgrade a desktop application to web / mobile
    - Improve Configuration and User Settings Management
    - Add International Localization
    - Improve resource files handling
    - Adopt a standard Composite Application Framework
    - Adopt a standard System Architecture Framework (e.g., Enterprise Library)
    - Implement new UI framework (replace VB6 forms with WPF)
    - Implement distributed computing using (replace COM+ with WCF)
    - Separate of architecture layers (MVC)
    - Move from embedded SQL to Stored Procedures
    - Implement an Object-Relational data persistence design
    - Add Tracing, Diagnostics, and Usage Monitoring
    - Add Performance Monitoring
    - Add Performance and Scalability improvements
    - Add Readability and Coding Style improvements

#### 4.3.3.3 SDLC Process Changes

- Elaborate on SDLC improvements that are in scope for this project.
  Possible examples are the following:
  - Implement Automated Code reviews,
  - Implement Automated Unit tests
  - Implement Automated UI tests
  - Implement Continuous Integration

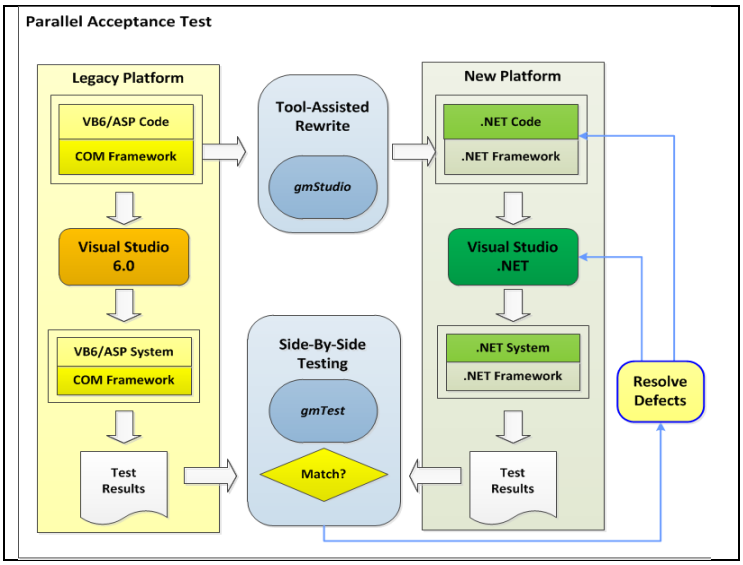
## 4.4 Verification

Testing and verifying each upgraded application is a critical part of every modernization effort.

A key consideration for modernization projects is that most requirements begin simply as the need to produce **functionally equivalent results**. Verifying this equivalence is often done using side-by-side or parallel testing and is illustrated in the figure shown here.

The biggest challenge with this type of testing is that expert knowledge and labor are needed to describe how to perform operations with the legacy application. These operating instructions must be organized, valid, and detailed to be used to generate expected legacy results for comparison to new results.

The functional requirement outline is the foundation for developing and organizing the operating instructions.




*Note: Great Migrations offers a UI testing IDE and execution engine that wraps MS Coded UI Test (as distributed with VS2013 Premium). We call this gmTest. gmTest can help the test team develop and use readable test scripts that can automated UI testing during and after the upgrade project.*

Please elaborate on your testing requirements and expectations for this project.

- Describe the overall QA process including team capability and availability to perform testing tasks.
- Describe your expectations for the vendor's role in the verification process.
- Describe your ability to setup and maintain parallel test environments for comparing legacy and source systems.
- Describe the performance and availability of the network, workstations, and servers in the test environments.
- List and qualify all test cases -- see functional requirements outline
- List desired automated UI testing requirements – see functional requirements outline
- List desired automated unit testing requirements – see member analysis
- List desired performance and scalability testing requirements.

## 4.5 Organizational Plan

### 4.5.1 Staffing Model

The following roles are typically needed on a legacy modernization project:

**Project Manager** – based on knowledge of the overall work plan and external factors that might constrain and impact the modernization effort, these resources will coordinate and direct of the work, facilitate approval of deliverables, and ensure successful closure of the project.

**System Architect** -- based on knowledge of the source application and the desired architecture design, standards, and constraints, this resource is will help formulate and approve technical decisions, articulate a clear and consistent vision of the desired system/application architecture, and the identify and close gaps between current and desired technical design.

**Modernization Specialist** -- based on knowledge of gmStudio and gmStudio upgrade solution development, these resources will help develop technical requirements and implement, improve, and maintain the upgrade solution over the course of the project.

**Application Developer** -- based on a strong working knowledge of the existing code and of .NET technology, these resources will facilitate problem identification, analysis and resolution and provide technical requirements and other resources in the upgrade solution.

**Software Configuration Manager** -- based on knowledge of the legacy source code structure and version control and a detailed understanding of standard builds and deployments across environments, these resources will provide requirements for and support implementation of builds and deployments of the new system in for the project.

**Network Engineer** -- based on knowledge of the access security, networking, workstations, and servers in the project environment these resources will provide infrastructure and network access support for the project.

**Database Administrator** -- based on knowledge of the databases and database servers used by the legacy and to be used by the new systems these resources will provide database administration support for the project.

**Quality Assurance Team** -- based on detailed knowledge of the application functionality and the processes to verify it, these resources will provide the test case/date documentation, and test scripts needed to setup and conduct comprehensive regression testing of the application.

- The members of the IT organization that currently maintains and supports the legacy application may be ideally suited to contribute to the modernization project if they are available. Please indicate if members of this group will be available to participate in the upgrade effort by serving in the roles described above.
- Describe the expected staffing model for the project. For each member of the project team, provide the following attributes:
    - Name: may be generic (developer1, QA1, PM)
    - Organization: client, vendor
    - Group: user groups, IT groups, vendors
    - Role
    - Hourly Rate or cost for estimating purposes (if applicable)
    - Experience in role
    - Responsibilities / Tasks
    - Time Allocation over the course of the project
    - Expected Physical Location
- Provide a hierarchical organization chart for the desired project team.
- Describe the development organization supporting the legacy application (in particular number of years working with legacy application and number of years working with .NET).

## 4.5.2 Communication Plan

### 4.5.2.1 Team Communications

The expectations for ongoing communication and knowledge management vary from project to project. GM typically sets up a secure WIKI for general document management, and an issue tracker for work management. We use the Atlassian products: Confluence WIKI and JIRA Issue tracker. In addition, GM will typically plan for periodic project status meetings with the project management team and periodic technical meetings with development and technical team; separate meetings for the quality team are also common. The frequency, duration and format of meetings will vary.

- Describe your requirements for ongoing project meetings.
- Describe any specific collaboration groupware for project management you wish to use?
- Describe any specific issue tracking groupware you wish to use?
- Who will attend project management meetings?
- Who will attend technical management meetings?
- Who will attend quality management meetings?

### 4.5.2.2 Project Documentation

Each organization has different conventions and standards for project and technical documentation. A typical project will have overall project planning and estimation documents as well as a variety of technical design, and testing-related documents. Furthermore, gmStudio is driven by various rules files, and many details of the upgrade solution will be documented in those files.

- Describe your requirements for project documentation.
- Provide standard project documentation templates if available.
- What is your process for organizing, versioning, and accessing project documentation?

### 4.5.3 Manual / Automated Work Balance

When using gmStudio, the vast majority of the code upgrade is automated and repeatable; however, there may be specific sections of code that are more efficiently upgraded by hand. The products of this manual work may be integrated with the rules that drive the automated processing making those results repeatable and automated as well. For some parts of the application, the decision to use manual rework and development may be known in advance. When this is the case, the plans for manual coding should be documented and considered when planning and estimating the project.

- Describe the parts of the application you intend to create from scratch or replace with hand-coded or third-party binary components. This information may be added to the functional requirements outline or the source structure model where appropriate.

### 4.5.4 Supporting Maintenance Releases

Because the tool-assisted rewrite produces a repeatable solution, it may be applied to a changing legacy system with very little disruption. Consequently, a legacy source code freeze is not required during the project. The Tool-Assisted Rewrite methodology allows for the source code baseline to be updated when the legacy application has a major release to production – typically once every 2-3 months. Some work is needed to set up and verify the source and the upgrade solution each time a new source code baseline is introduced, so it makes sense to describe the legacy application release forecast during planning.

- Describe the legacy system release processes including frequency of releases, testing, and transition.
- List the expected timing, nature, and size of legacy system releases that will occur during the modernization project.
- List any major technical changes expected during the upgrade project (i.e., OS upgrades, technical dependency changes, integration changes, DB server upgrades, etc.)

### 4.5.5 Transition Plan

Some organizations prefer to upgrade large systems or collection of applications incrementally, as a series of separate independent upgrades. A typical scenario is to do the smaller independent parts first to gain experience and refine processes before attempting huge, complex systems. This “ramp up” strategy can reduce risk and improve efficiency as lessons learned are accumulated and integrated into the solution for each successive effort. Optionally, separate upgrade efforts may be done in parallel. Working in parallel can reduce the overall duration of the program, but requires more staff and more coordination if lessons learned are to be shared across teams.

- Describe how/if the systems to be upgraded may be divided into parts that can be upgraded independently or incrementally.
- Describe the general timing and expectations for how the new systems will be phased into production.

### 4.5.6 Closure Criteria

One of the most critical aspects of a modernization project is final acceptance and closure process.

- List your acceptance criteria and required deliverables.
- Describe the approval process for declaring the project closed and successful.