UNIX Code Migration Guide

UNIX Application Migration Guide

patterns & practices
proven practices for predictable results

Chapter 3: The Migration Process

Larry Twork, Larry Mead, Bill Howison, JD Hicks, Lew Brodnax, Jim McMicking, Raju Sakthivel, David Holder, Jon Collins, Bill Loeffler
Microsoft Corporation

October 2002

Applies to:
Microsoft® Windows®
UNIX applications

Summary: Chapter 3: The Migration Process takes a detailed look at the overall migration process and works through the structure that any migration requires. (20 printed pages)

Contents

Introduction

Migrations of UNIX applications to Microsoft® Windows® can vary significantly in size and scope.

Every organization discovers differences in the impact a migration has on the application and its development process, and on the user environment and support structures. Despite the many differences among migrations, the basic migration process is the same for every organization.

This chapter contains a high-level description of each stage of the migration process. The details of each stage are covered in later chapters.

Purpose of This Chapter

This chapter helps you understand the migration process as a whole. After reading this chapter, you will be able to:

- Understand what you need to plan and prepare for the migration
- Clarify key decisions involved in the migration
- Define procedural steps for the migration
- Create a project-plan outline
- Predict the impact of the migration on your organization

Audience

The information contained in this chapter is useful for everyone who is involved in the migration process. However, it is particularly relevant to managers and project managers who are responsible for the decision and planning processes.

Scope

Every organization has its own structure, project methodology and decision-making process; this chapter does not tell you to change these. Instead it provides the information that you need to carry out a migration within your own system and methodology.

Background and Assumptions

The information presented in this chapter is based on the assumption that you have already made the most important decision—that is, to migrate to Windows.

Organizational Structure and the Development Life Cycle

Before reviewing the migration process, it is helpful to look at the different parts of an organization that are involved in the development, implementation and life of the applications. Because all migration projects are different, this chapter uses a generic development and production environment. For information about more specific environment types, the issues they raise and how to work through those issues, see Chapters 7 and 10.

Figure 1 shows a typical organization, including some of the departments that are involved in an application's life cycle.

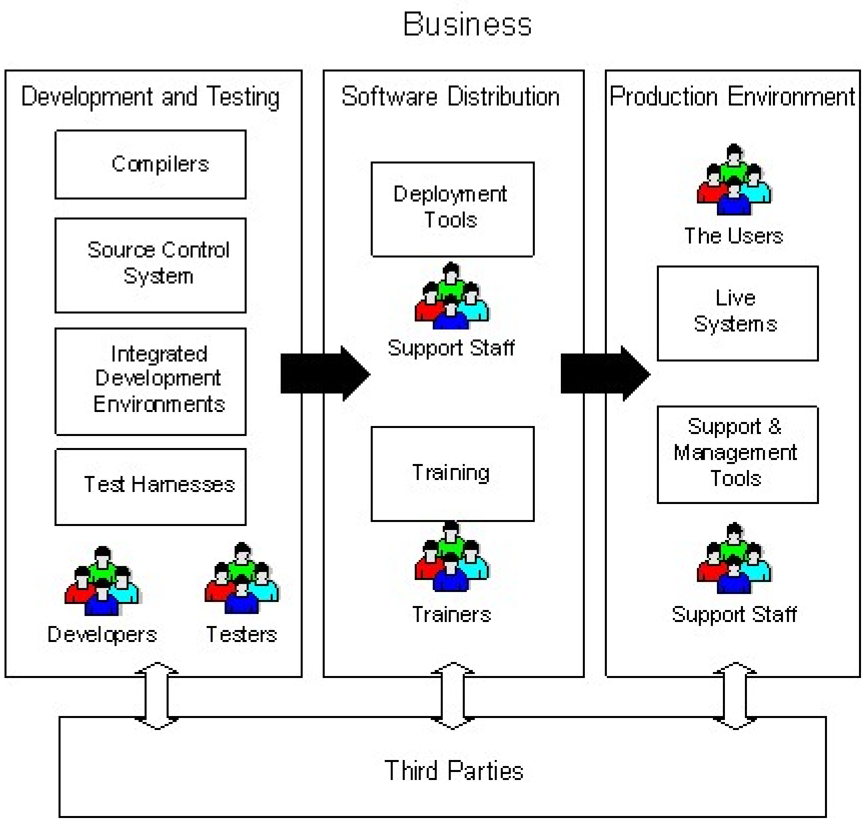

Figure 1. A conceptual organization and its application life cycle

Your migration will have an effect on all of the departments and systems shown in this figure. Later sections of this chapter describe how you can estimate the impact on your business.

Overview of the Migration Process

A migration process can be organized many different ways. No two UNIX-to-Windows migrations will be the same. For convenience of example, this chapter structures the process as shown in Figure 2. This structure includes all the steps necessary in the UNIX-to-Windows migration. Information in this guide follows the structure shown here.

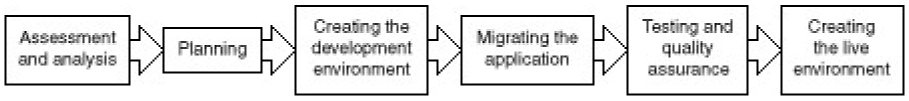
Assessment and analysis
Planning
Creating the development environment
Migrating the application
Testing and quality assurance
Creating the live environment

Figure 2. The generic migration process

The following sections provide a high-level description of each of these steps to help you make initial decisions and begin planning the migration.

Assessment and Analysis

The first step in your UNIX-to-Windows migration is a detailed assessment of all aspects of the application. This begins with the collation of information about the application, its context and its user base that can be applied to an evaluation of your application's business and technical objectives. You can then assess the application and the environment in which it operates, including its development environment, user environment and support environment.

You should consider both technological and cultural factors throughout the assessment and document any issues that you discover. These can be incorporated in the design of the migration plans and schedules.

You can accomplish only a high-level definition of the application porting at this point. Later on in the migration process, you can develop initial prototypes of the application to help you decide whether a final port, a rewrite, the purchase of a software package or some combination of approaches is the best way to conduct the migration.

The results of this assessment can help you determine the best approach to the application migration and define the migration strategy. You can review the migration strategy in conjunction with the issues that you documented during the analysis to ensure that they have been handled properly.

A full description of the assessment and analysis process is presented in Chapter 4, Assessment and Analysis.

Gathering Application Information

To determine the objectives of the application that you want to migrate from UNIX to Microsoft Windows, you need an in-depth understanding of the application, how it is used and its infrastructure and support environment. During this first part of the assessment, you gather all of the business, technical, operational and other information that can help you determine the appropriate application-migration approach and inform your decisions during the migration process.

The following sources can provide information that will help you understand the objectives of the migration:

- The application
- The application's documentation
- The current infrastructure
- The development environment
- The user base

The information you collect from these sources will allow you to develop a migration strategy, as shown in Figure 3.

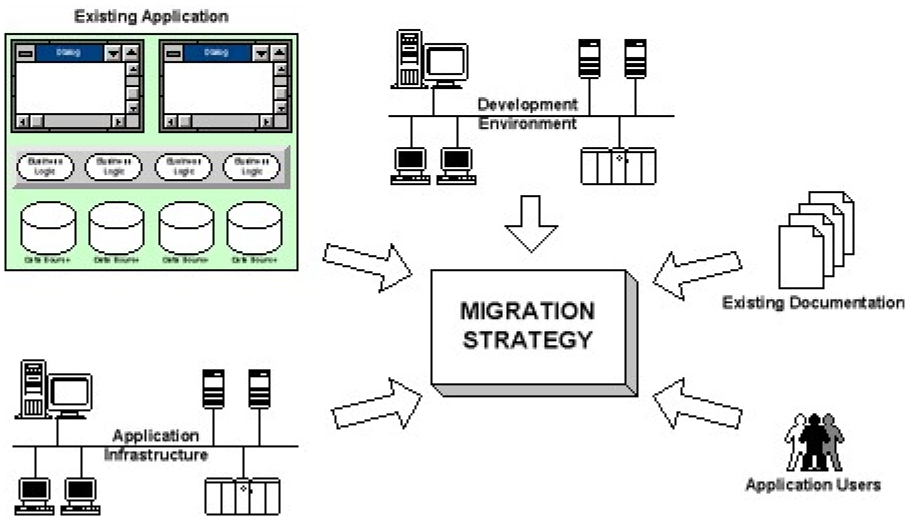

Figure 3. The migration strategy

The following list contains some questions that can yield the information you need to make appropriate migration decisions (a more complete list is provided in Chapter 4, Assessment and Analysis):

- What does the application do?
- How is the application built?
- How does the application run?
- How is the application accessed?
- What does the user expect from the application?

Evaluating Business and Technical Objectives

When you evaluate the business and technical objectives, you establish a foundation for the migration. The evaluation should meet three goals:

- To understand the business objectives for the application
- To understand the technical objectives for the application
- To understand the objectives of the migration

In many cases, the objective of a migration is to minimize redevelopment costs. Therefore, the evaluation does not have to be an exhaustive study, as would be required for a new application.

Business objectives

Application requirements change over time and it is not unusual for older applications to no longer meet the users' needs. You should verify that the application functionality is relevant to current business needs.

In addition, you should determine how the organization intends to use the application, because the use has significant bearing on the migration. For example, the plans could be to:

- Significantly enhance the application
- Make minimal changes to the application
- Replace the application within a foreseeable timeframe

- Use the application as a basis for other applications
- Maintain common code across the UNIX and Windows platforms

Technical objectives

During the assessment, you should confirm that you understand the technical objectives of the application. In other words, how should the application be built and how should it run? Technical objectives include:

- The service characteristics of the application (for example, scalability and availability)
- The technical constraints on the application (for example, architectural restrictions or the need for specific protocols)
- The portability requirements, as in whether the application has to run on multiple platforms
- The integration constraints, as defined by the number and types of other applications and services with which the application interfaces

Understanding the technical objectives helps you know how complex the application migration has to be and what technical issues must be addressed during the migration process.

Migration objectives

The migration process is driven by a number of key business factors. The motivation for migrating to a Windows environment must be clear. All parties involved should reach a consensus on the migration goals, because the goals have a significant effect on the decisions involved in planning a UNIX-to-Windows migration.

The following examples are some reasons for migrating to Windows:

- To reduce the total cost of ownership by deploying a homogeneous computing environment that is based on Microsoft Windows
- To improve integration with other applications running on Windows-based computers and servers
- To enhance the application by implementing usability features that are based on the Microsoft Windows look and feel
- To provide a greater range of hardware options that offer better choice and scalability
- To integrate the application into a mobile computing environment
- To accommodate dependencies on essential third-party products that are also being ported to Windows
- To improve support by concentrating skills on the Windows platform
- To create a UNIX-style development environment after the migration (for example, to accommodate developers who know UNIX but not Windows)

You should develop a clear idea of your own objectives because they affect the migration strategy that you choose. For example, a migration to achieve usability enhancements will have different requirements from one to provide improved integration.

Evaluating the Application

Before planning the migration, gather information about the application architecture, interface and communication mechanisms; its internal structure; its build environment; its mode of deployment; and other supplementary information. Much of this information can be found in existing documentation.

Additional application-specific information can be useful:

- The history, including the porting history (for example, has the application already been ported to UNIX environments such as BSD or System V)
- The use of third-party development tools and libraries, including cross-platform facilities
- The standards that the application supports, particularly proprietary standards
- The management and support scenarios (for example, the use of monitoring tools and remote shells)

The evaluation can be supported with code-analysis tools. Code analysis is performed for many reasons, including:

- Portability analysis (for example, to determine UNIX API dependencies)
- Performance analysis (for example, to discover which portions of the code are accessed most often)
- Conformance analysis (for example, to determine ANSI C/C++ conformance)

The results of this analysis help to determine the type and complexity of the application that is to be ported.

Note that a complete description of how to evaluate an application is provided in Chapter 4, Assessment and Analysis.

Defining the Migration Strategy

The evaluation of the application and its context is a significant factor in deciding the form of the migration. Additional strategic decisions are necessary, including these:

- What is the target environment?
- Do you need to maintain common code across UNIX and Windows platforms?

These decisions are summarized and documented in the migration strategy. You should deliver the migration strategy along with an overall infrastructure conceptual design that encompasses all of the migrated application's dependent infrastructure requirements, including the following aspects:

- UNIX and Windows resource sharing (for example, file sharing)
- Security and user-account synchronization (for example, integration and interoperability with Active Directory and Network Information System)
- System and application management and support (for example, updates, packaging, remote support)
- System scalability and performance
- Documentation migration

The migration strategy can consist of a quick port, a full port, a rewrite, coexistence of the old and the migrated application or some combination of approaches. These strategies are presented in more detail in Chapter 4, Assessment and Analysis.

Quick-port strategy

Interix was introduced in Chapters 1 and 2 as a UNIX-conformant environment that runs on the

Windows platform. When the results of the code analysis indicate that a port to Interix is the most appropriate migration strategy, a quick-port migration is the best approach. This is achieved by obtaining the source (for example, the archive) and installing it in the Interix development environment, modifying the configuration scripts and Makefiles, and building the application.

The ported application may be successful immediately or may require modifications to the configuration scripts or Makefiles to account for the new hardware platform, the target operating systems and local configuration information.

You may not be able to determine whether or not a quick port is possible without actually conducting the port. The port may fail because:

- The application proves too complex to be ported quickly.
- Some parts of the application have to be rewritten for the port to be successful.
- UNIX-environment dependence remains that cannot be accommodated by a quick port.
- The quick port uncovers some new issues or shows that your assumptions were incorrect.

Often, you can attempt a quick port as the first stage of the migration process. Then, if it does not work you can use the results of the quick port as a basis for a full port or a rewrite.

Full port strategy

During a full port, the application is migrated with the minimum necessary changes to the source code and by using UNIX-compatible libraries and utilities that are available on the Windows platform (and that can include Interix). Unlike the quick port, however, significant changes may be required to enable the code to run on the new platform. This can happen when the original code is not fully standards-compliant or when code elements (for example, device drivers) are specific to the original system.

The following are reasons for choosing to directly port an application to Windows:

- Rewriting the code from scratch is seen as too costly or time consuming.
- A large amount of source code exists.
- Cross-platform support for application is required (such that a single base of source code is maintained).
- The application makes use of a large number of UNIX APIs.
- You want to retain the current look and feel of the UNIX application.
- The integration with Windows applications can be achieved without a significant rewrite.

As long as the application functionality is tightly controlled, the full port strategy reduces the risk of negatively altering the application, because you preserve the business logic, retain the same user interface, reduce the need to for new documentation and require virtually no retraining of end-users.

Rewrite strategy

A rewrite strategy is appropriate when the costs of porting the application may outweigh the benefits or when specific application qualities (such as performance or scalability) require that code be written specifically for the Windows platform. Rewriting your application from scratch has a number of significant advantages:

- In situations where a single platform is being used, a rewrite results in the maximum use of the

advanced features of the native Windows environment.
- A rewrite offers the best performance in a native Windows environment.

You may choose to rewrite an application to the Microsoft Win32® application-programming interface for the following reasons:

- Win32 features are required in new versions of the application.
- Third-party applications are now available in the Win32 environment.
- You intend to standardize your development environment on Microsoft Visual Studio®.
- Rewriting offers the best possible integration with other Windows applications.
- The application's performance is critical; therefore, it should run as a native Windows application in the Windows environment.
- Cost and time are not constraints.

A rewrite is potentially the most costly and time-consuming option. The solution based on this strategy requires a thorough analysis of the UNIX application functionality in advance of a Win32-based redesign of the application architecture so that the rewrite of the UNIX code takes full advantage of the Windows application platform. This strategy has the greatest potential risk, but it also can produce the best results.

Coexistence

For coexistence, you port or rewrite the application, but retain the original application after deploying the new application. There are a number of reasons to choose this approach:

- The two applications (source and migrated) are to be run in parallel to minimize risk.
- A new set of users or customers will use the ported application, leaving the original users unaffected.
- A staged migration is planned between the original and the migrated application.

When you plan to maintain coexistent applications, parallel development and build environments should be maintained to support all the platforms on which the application will run (UNIX and Windows). This may require a cross-platform source control system, such as Revision Control System (RCS), Source Code Control System (SCCS), Concurrent Version System (CVS) or Program Version Control System (PVCS). The native Windows source-control system is Microsoft Visual SourceSafe®, which supports the cross-platform Source Code Control Interface (SCCI) API. Interix supports the Revision Control System.

Strategy combinations

You may decide that it is appropriate to mix strategies. This often happens with large applications where certain portions can be rewritten and others ported. You may choose to port and then rewrite the application to reduce the overall time requirements on the project or to accommodate short-term time or budget constraints. For example, the migration could begin as a port and move forward as an incremental rewrite based on the integration or evolution of portions of the application. As in a port strategy, a common source-code base across UNIX and Windows can be retained. This strategy is similar to the way in which UNIX software is ported between UNIX platforms, because it allows developers to take advantage of platform-specific functionality.

To ensure that you have selected the most appropriate strategy, you should perform a cost-benefit analysis that weighs the benefits of the strategy against its costs. Part of the analysis should include the option not to migrate, but to buy an off-the-shelf Windows-based solution instead. The following are important business considerations:

- Is the migration necessary?
- What is the business risk of migrating?
- Is the business impact acceptable?
- Are the benefits worth the cost?

These questions can be used as a basis for determining the benefits, costs and risks that may be associated with the migration. The analysis itself should be proportionate to the scale of the migration. Cost-benefit analysis techniques are presented in Chapter 4, Assessment and Analysis.

Planning the Migration

When you have completed the assessment and made a decision about the strategy for the migration, you can plan the necessary steps for the migration to begin. This planning stage involves the definition of objectives and schedules for the migration so that you can find the appropriate resources and allocate tasks as necessary.

The activities involved in planning can be grouped into starting the project, developing a catalog of issues and drafting a project plan. The project plan should be as accurate as possible at this point in the mitigation project. However, do not attempt to produce a plan that is too detailed, because the scope of the migration may change after porting or rework activities are underway.

Planning is covered in more detail in Chapter 5, Planning the Migration.

Starting the Migration Project

The migration project has already started, in that assessment and analysis activities are ongoing. When these are complete, initiate a plan for the migration itself. You can then establish key criteria for the mitigation and may actually discover that a single assessment and analysis process can accommodate multiple migration projects.

The following are essential start-up activities:

- **Establish the project scope**

  Make sure that the requirements for the migration are well defined and bounded. Otherwise other application elements may be drawn into the migration, which can create a migration-project overrun

- **Establish the resources**

  Determine the available resource levels for the migration project. Because it is unlikely that infinite resources are available, migration activities can be limited to the available personnel and budget resources.

- **Document the success criteria**

Arrive at a consensus about the project success criteria with the project stakeholder or user representative to ensure a basis for agreement about when the project is complete

- **Determine the preliminary schedule**

  It may not be possible to determine a definitive schedule until after the plan has been produced, and, the schedule may need to change after the project is underway. However, preliminary deadlines are useful in the planning process.

Developing a Catalog of Issues

Throughout the assessment stage, you document issues that can or should influence the migration. The issues may be schedule related to business, technical, application or infrastructure aspects of the application or the overall mitigation project. Before you can create a realistic project plan, you must determine whether any of the issues pose a serious risk to the project, and what can be done to minimize the possibility of those risks.

You may want to investigate the following issues:

- **Business issues**
  - The impact of the change on the end-users
  - The shortfall between the application functionality and user need
  - The distribution of the user base
- **Technical issues**
  - The required experience to perform the migration
  - The technical complexity of the solution
  - The security requirements
  - The application deployment
  - The dependency or effect on other applications
- **Application issues**
  - The quality of the code and scripts
  - The quality of the development environment (for example, source control, configuration, build, debugging and bug tracking)
  - The required service levels for the application
  - The standards conformance requirements
- **Infrastructure issues**
  - The hardware performance
  - Interoperability and integration

A more complete list of potential issues is presented in Chapter 5, Planning the Migration.

Each issue can create a number of risks in the project. Those risks can be quantified in terms of probability and impact. You should determine mitigation actions for each risk to ensure that the risk is minimized or removed.

Producing a Project Plan

The migration project plan should take into account the type of migration to be performed, the issues that are identified (and the mitigation for the issues), the funding provided and the availability and skills of the people who are assigned to the migration project. The project plan is based on the project stages described in this document, and the activities involved in each stage vary according to the chosen migration strategy.

Creating the Development Environment

Create the development environment during the first stage of the migration. The development environment for a migration can be more complex than for an original application. For example, many elements of the development environment itself (such as configuration scripts and Makefiles) may have to be ported. In addition, you will need to obtain the application code that is to be ported and store it in the new development context.

The development environment is covered in more detail in Chapter 7, Creating the Development Environment.

Installing the Development Environment

To migrate the application, you must provide a development environment in which the application can be imported, built, tested and maintained. Before you start the application migration, identify the following elements of the Windows development environment:

- The source-code control system that is used during and after the application migration
- A problem-tracking system
- The appropriate assessment and code-profiling tools
- The performance analysis tools
- The testing and debugging tools
- The development environments and software development kits (SDKs)
- The third-party libraries

You should not consider the development environment as a short-term configuration for the migration. Rather, consider it to be part of a long-term maintenance strategy for the migrated application.

Populating the Development Environment

You populate the development environment by obtaining the original source from the source system as a code hierarchy, as an export from a source-control system or as an archive.

Some code may have been generated on the source system. You need to decide whether the generators will continue to be used or whether the generated code will serve as a baseline for subsequent developments.

Configuring the Development Environment

Configuring the development environment includes creating or configuring the build process. Some build-process testing is possible, but some (for example, determining whether the code compiles and links) is part of the port or rewrite itself.

Migrating the Application

You must specify in detail the target environment so that your developers can complete the application migration. You have to provide clear guidance about the techniques to be used in changing from a UNIX approach to a Windows approach. You also have to define which of the Windows-specific features you want to use in your application.

The Target Environment

You determined your migration strategy during the assessment and analysis. You should now know whether you are porting to:

- The Win32 environment and API
- The Interix environment and API
- A third-party environment and API (for example, a cross-platform environment like Roguewave)

Your choice of target environment determines the appropriate API and the available libraries.

Completing the Pre-Migration Changes

After you have determined the target environment you may be able to implement standards and procedures that make the code easier to migrate. These include changing code so that it continuously works on the source platform, but also works on the target platform. For example, if you are migrating to the Interix environment, then you can ensure that the application code conforms to the appropriate standards (for example, POSIX) before the migration.

Specifying the Coding Strategies

Your code analysis helped you recognize the differences between the current code and the target environment. You now have to specify a standard approach to each of the differences and determine a way to map from one to the other.

A full list of expected differences is presented in Chapter 7, Creating the Development Environment. The following are some examples:

- File names, access mechanisms and structure
- Interprocess communication
- Networking and remote access
- User interface and screen management
- Conversion of daemons to services

Converting the Code

You can now start migrating the code and building the application on Windows. If you build the code quickly, you can also create a functioning prototype that you can use to test the effectiveness of your strategy. You may find it useful to migrate a part of the application to assess the success of your strategy.

Migrating the Scripts

UNIX application and development environments commonly use scripts. You may have to migrate at least some of these scripts to the target environment. As with the source code, you have the choice of rewriting the scripts or porting them. In many cases, versions of the scripting languages used in UNIX are also available in Windows, making it relatively straightforward to carry over the scripts. However, the scripts sometimes have to be rewritten using one of the scripting languages that are available in Windows.

Testing and Quality Assurance

Testing a migrated application is not always as straightforward as testing an application that you have defined and built yourself. When you port an application, you do not understand its functionality the way you would if you had written the program. Typically, you do not have time to learn the application's internal architecture and design (this time constraint may be part of the reason a porting strategy was selected).

Despite some complexities, the testing activities are similar to those used in a homegrown application. First, write a test plan that describes test objectives and a plan of action for meeting them. Next, write tests that provide adequate coverage of the testing objectives. Finally, complete the tests and analyze the results to ensure that the application migration has been a success.

Testing is covered in greater detail in Chapter 12, Testing and Quality Assurance.

Writing a Test Plan

The test plan includes a testing strategy that specifies the areas to be tested and describes the resources (both hardware and personnel) that the test team requires in order to do its job.

The testing strategy defines the overall approach to testing and takes the following into account:

- The type of application to be tested (for example, user facing or process control) as well as the main functional aspects of the application
- Key quality criteria for the application (for example, whether it requires being transaction-intensive, highly reliable or scalable)

Based on these factors, the test plan proposes how the application should be tested. For example, the plan could stipulate that the application requires mainly user-facing tests or that it specifically requires loading tests.

The strategy should also take into account any testing requirements and facilities already in place or defined to enable testing of the migrated application. The following are some questions to ask:

- Does the application include test application(s) or script(s) that can be used to perform functional or performance-verification tests?
- Does the configuration and build environment build the test application(s) along with the target application?
- How portable to the Windows environment are the test application(s) or script(s)?

The type and amount of testing depends on the application that is being ported, the strategy chosen for the migration and the availability of existing information and resources. If no testing applications or documentation exists, a side-by-side comparison of functionality of the UNIX and ported applications is the only alternative.

Writing the Tests

The following are some applicable test categories:

- **Coverage testing** (used primarily during the development phase)
  - Test every feature of the migrated application (a functional test)
  - Test the code base of the application (check-in, build verification and regression tests)
- **Usage testing** (used primarily during the stabilizing phase)
  - Test to discover the sections of code on which the application spends most of its execution time.
- **Integration testing** (used to verify the correct interactions between the application components and other systems).
  - Test the synchronization and coordination between components (for example, the way that UNIX and Windows workstations use the migrated application to access data).
  - Test the data integrity, especially when the application requires intensive file record-locking.
  - Test the features and the usage scenarios.

Tests should also be planned for any configuration, runtime or deployment scripts.

Running the Tests

After the tests are written and the application build is successful, you can actually run the program. Did it work? If so, you can continue to test its functionality and performance by using the tests that have been defined for the application.

If the application is packaged with its own suite of testing applications or scripts, perform a portability analysis of the test suite by using the same tools and methodology that you used with the target application. If the application does not have a testing suite, use a symbolic debugger as the main testing tool if the application fails.

If both the UNIX and ported programs have testing applications or scripts, and the testing applications or scripts have also been ported, then the tests can be performed on both the original UNIX and the ported applications. The comparison of the test results can be automated by using tools like diff to compare the functional output line by line.

Even if no testing applications or scripts exist, the UNIX application may be bundled with documentation that describes its input-output functionality. You can then compare the test results to this documentation.

Creating the Live Environment

Most migration projects change the application's live environment. This section describes the areas that might be affected by your migration.

The live environment is discussed in greater detail in Chapter 13, "Creating the Live Environment."

Hardware and Software Infrastructure

Your new application requires Windows platforms on which it can operate. The rollout of Windows platforms is beyond the scope of this guide, but is covered extensively in other Microsoft guides.

Purchase and configure your Windows servers and clients, as well as other infrastructure elements such as networking equipment, as appropriate for the application.

> **Note** For further information about migrating your live environment to the Windows platform, please see the guides at the Microsoft TechNet Web site. You can also find helpful information at the Web site Migrating to Windows from UNIX and Linux.

Interoperation

Most migration processes require interoperability between the old and new environments. The length of time for which interoperability is required differs significantly for different types of migrations.

To accommodate interoperability, you can consider these strategies:

- **Create a Windows-only environment**

  If the project is low-risk, you may be able to run the Windows and UNIX environment in parallel only for the time it takes for the users to make the transition to the migrated application.

- **Create a separate Windows environment alongside the UNIX environment**

  This approach is suitable for short-term parallel running or when the application has both UNIX and Windows modules.

- **Create an integrated cross-platform environment**

  This approach is ideal when the two environments will coexist for a longer period time.

You can use a variety of tools to enable interoperation between UNIX and Windows. In the live environment, users and support staff are most often interested in gaining access to both systems and sharing data (typically files) between the two environments.

Windows and UNIX interoperation

Users probably require four main areas of interoperation in your live environment:

- Client connectivity
- File sharing
- Resource sharing
- Security and user authentication

Microsoft and third-party sources offer a wide range of products and tools that assist in each of these areas. Research these products after you determine your specific requirements.

> **Note** Microsoft's Windows operating systems include basic interoperability with UNIX over TCP/IP as a part of Microsoft® Services for UNIX. In addition, Interix provides additional tools (such as X-Windows clients) that assist further integration.

Deploying the Migrated Application

The deployment of the migrated application takes place in two phases: the initial deployment and an ongoing deployment of upgrades and fixes. Many solutions exist for application deployment; some are UNIX based and some Windows based.

The following approaches can be used to deploy the migrated application:

- **Native Windows deployment**

  A native deployment uses Microsoft Installer, the standard software distribution and installation method for Microsoft Windows 2000® and Microsoft Windows XP. This approach requires software that is packaged in MSI (Microsoft Installer) format.

- **Manual deployment**

  In manual deployment, the application is installed individually on each computer. This approach is appropriate for applications that are infrequently deployed to a small number of computers.

- **Automatic (custom) deployment**

  This approach makes use of a scripting tool or third-party product to deploy the software automatically.

- **Package deployment**

  This approach uses a package that can be distributed to remote client computers (for example, by using Windows Installer) and can then be deployed by remote users or customers.

UNIX systems have a wide variety of software management and distribution tools, but the tools are often incompatible. Many UNIX environments make extensive use of bespoke scripts to deploy applications. You have two main choices:

- Use a new Windows solution.
- Migrate the software distribution to Interix or similar environment. The migration options and process will be similar to migrating the software application.

Support and Maintenance

An appropriate level of support and maintenance is key to the success of your migration project. Your support staff and systems have to be ready for the new environment. Plan to migrate the appropriate support tools and monitoring systems and retrain your support staff.

User Training

The user training that is required after the UNIX-to-Windows migration depends on the scale of the difference between the application's UNIX user interface and the Windows user interface. After a port to the Interix environment, the user base may experience no change. With a port to the Windows Win32 environment, the users may require some training.

You should assess the training needs of individual users before you implement the live environment, so that user training can be planned appropriately.

Summary

This chapter broadly describes the migration process. Figure 4 summarizes the basic stages of a migration process and shows the typical deliverables for each stage.

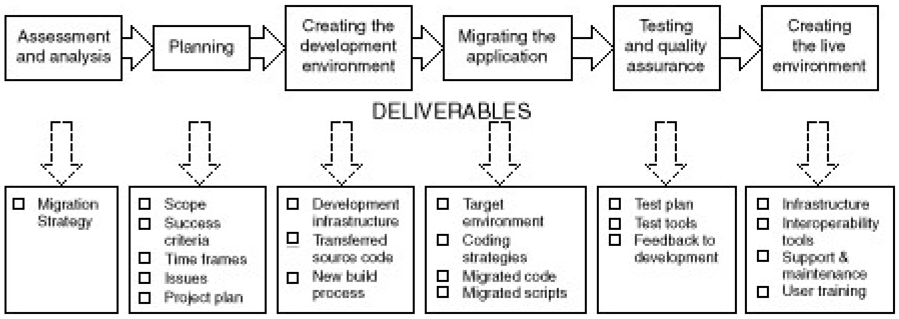

Figure 4. The migration process and key deliverables

You can now begin the migration process by initiating the assessment and analysis stage, which is described in the next chapter.

patterns & practices
proven practices for predictable results

Send feedback to Microsoft