UNIX Code Migration Guide

UNIX Application Migration Guide

Chapter 5: Planning the Migration

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

The *patterns & practices* team has decided to archive this content to allow us to streamline our latest content offerings on our main site and keep it focused on the newest, most relevant content. However, we will continue to make this content available because it is still of interest to some of our users.
We offer this content as-is, without warranty that it is still technically accurate as some of the material is undoubtedly outdated. Note that the content may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

Summary: Chapter 5: Planning the Migration addresses the final touches on your migration plan, combining the high-level migration process from Chapter 3 with the detailed assessment of your application in Chapter 4. Upon completion, you will have the complete documented framework for your migration. (19 printed pages)

Contents

Introduction

Although the migration project has already started, it is not until the analysis and assessment activities have finished that you are able to judge the scale and scope of the migration. At this stage you are able to:

- Initiate the migration project, by confirming its intentions and deliverables.
- Catalog and analyze issues and risks that may jeopardize the migration.
- Produce plans and schedules for the coming stages of the project.
- Identify, agree upon and allocate resources to the project.

This section presents a project framework based on the Microsoft® Solutions Framework (MSF). This framework is a methodology for running software projects and illustrates how software development cycles can be used to create quality software on time and within budget.

Every software development project is part of a larger software development lifecycle. During the lifecycle, all of the activities necessary for creating, shipping or deploying software repeat with each release. The repeated steps include specifying the project deliverables, implementing them, testing the resulting modules and creating documentation. The MSF process model is milestone-based, as is the traditional waterfall model, but it also includes the spiral model often promoted for Rapid Application Development (RAD).

The milestones specified in the process model in Chapter 3, The Migration Process, should be considered review and synchronization points rather than freeze points. The milestones give the team a tangible goal to work toward, as well as a time to make adjustments, if necessary. In the process model, a milestone concludes each phase. To promote objective measures, each milestone should be easily quantifiable and correspond to the completion and approval of a project deliverable.

Starting the Migration Project

Initiation of a migration project provides a starting point for subsequent activities. Ideally, the project would have been initiated right at its start, before assessment and analysis have taken place. However, it is only during these analysis activities that you can determine the full picture of what needs to be done. Even if a project start-up did take place before the assessment stage, it is essential to revisit startup activities following assessment and analysis.

Project startup enables you to:

- Explain and clarify the planned intentions and deliverables of the project, so that you can manage the expectations of the project stakeholders.
- Ensure that the requirements are well defined and bounded; otherwise, other application elements may be drawn into the migration causing it to overrun.
- Confirm objectives and timescales for the migration, so that you can manage expectations, recruit appropriate resources and allocate tasks as necessary.

It is quite possible that a number of migration projects are identified; for example, one for back-end databases and one for specific user-facing applications. In this case, the project startup enables each project to be defined and any dependencies to be identified so that the migrations can take place as independently as possible.

The end of the initiation phase is marked by the completion of the first milestone where the migration team and the customer agree on the overall vision and scope of the project. The vision is the complete view of what the solution can be—the refinement of the original "bright idea." On the other hand, the scope establishes restrictions on the vision to establish what can be accomplished within the constraints of the project. In a way, creating the vision and determining the scope are opposites, but both pieces are required for a successful project.

For migrations from UNIX to the Microsoft Windows® operating system, the vision is having an existing UNIX application work on the Windows platform. The application may include compiled application specific code and libraries that are used by the application. Many UNIX procedures also include scripts that tie parts of the application together into an overall process. Part of defining the vision is determining which parts of the overall process must be migrated now, and which can be done later.

Choosing a Project

The business case for the migration is based on output from assessment and analysis; this enables you to investigate migration options and decide on a suitable approach. The business case can now be used to agree on the goals for the migration, selecting from any alternatives that might still exist (for example, whether a package should be bought). This maps onto the envisioning phase of a software development project, where the team decides on the direction of the project (the vision) and determines its exact goals in the form of a vision/scope document. You can assign success criteria to each goal, agreeing on these with the project stakeholder or user representative to provide a basis for agreement when the project is complete.

The sponsor of the project begins this process by establishing a clear understanding of who the customer is and who the end users are. The customer pays for the migration and the end users use the ported application. The customer may be a person or persons in management who have an interest in the project, or the customer may be another group within the corporation that wants to see the results of the migration process. The end user may or may not be in the same organization as the customer. For example, the customer may be information technology management that is interested in cost savings while the end users may be engineers or help desk personnel.

There are a number of activities that make take place during the project, such as:

- Deploying tools and building a development and/or a pilot environment
- Testing the migration or porting techniques by selecting a large enough source base and a complex enough build environment
- Running a prototype or pilot to verify the migration approach, to test tools and procedures and to build confidence in the planned solution

Each activity has a bearing on overall success of the migration process. In particular, you should handle prototyping activities carefully. Prototypes set the tone and expectations for the rest of the migration. Because of this, it is very important that the experience be positive and successful for the organization. A prototype project should be initiated as a test case for the larger migration project—this enables you to affirm that the staff, techniques and procedures are all up to the task.

The vision/scope document ensures that everyone on the team shares and understands the goals of the project. This allows each team member to work effectively toward the project's successful completion. Each team member can make his or her own decisions without consulting other team members or slowing progress with time-consuming meetings. While it may seem like overkill to have a detailed project framework for what may amount to a very simple project, keep in mind that the project may set the stage for larger and more complicated efforts. Any tools, environments and procedures you use have a lifetime beyond the migration itself, because they must support the application after it is ported.

Confirming the Scope

A primary deliverable of the planning phase is the functional specification. This describes the

requirements to be met by the migrated application and the way that these requirements are to be achieved. This may be as simple as getting the code to compile and run or it may involve extensive modifications to the source. Because the specification provides a foundation to measure results against, the importance of documenting the operational details of the planned application cannot be overemphasized.

Creating the scope of the effort includes determining which pieces should be converted and the way that the conversion is to take place. Options include using migration tools to run the scripts as-is, converting to a platform-independent language such as Perl, and porting the working pieces to run in a Windows scripting or compiled language. The team must determine whether the application must be maintained on both platforms. If so, this somewhat restricts the choices and the ability to take advantage of Windows platform features.

The scope of your project ultimately depends on the resources you have available to complete it. Accordingly, there are a variety of ways that you can envision the scope of the migration, for example:

- Do you merely want to see if the applications can be installed and launched properly? Or do you want to check further for specific functionality and performance?
- Do you want a thorough investigation of any problems? Or does it suffice just to make note of them and move on?
- Do you want to test every application currently running on the UNIX platform? Or do you want to focus only on the top priority applications?

After the functional specification is compiled, you can review it against the vision/scope document for scope creep (that is, expansion of the scope). It is important to confirm that the specification does not go beyond the vision/scope that the team agreed to. Otherwise, the project is in danger of not being delivered within the time and budget constraints.

If the functional specification extends the vision/scope document, the teams must invoke change management procedures to determine whether the changes are required and if the project can be extended. Otherwise, after all of the teams agree on its content, the document should be signed by the team leaders.

Setting Expectations and Deliverables

Confirmation of the scope and the functional requirements on the migration project are only two facets of the initiation triangle; the third facet involves setting the expectations of the team and the customer. The actual deliverables and other less tangible results combine to make up the expectations for the project, the lack of which can doom a project to failure. When the customer or management is expecting one thing and the team delivers another, astonishment is usually the result. It is advisable to have all of the parties agree on what the project will and will not accomplish.

The best way to ensure realistic expectations from both sides is to set out the success criteria for each deliverable. You can do this by considering the functional requirements in the specification and defining a measurable criterion against which the requirement can be judged. These may be straightforward, such as "ensure message can be communicated between system A and system B," or more complex involving a series of steps. It may not be possible to determine criteria for every requirement—for example, scalability and availability requirements are notoriously difficult to gauge. However, you should have reasonably complete coverage of all of the requirements, and you should agree on these with the appropriate stakeholders.

The success criteria that you define should link back to the objectives set for the project. Perhaps your organization is experiencing lower returns on its information technology investments as well as rising administrative costs, or perhaps your personnel are tired of administering multiple platforms and want to simplify the environment. Whatever the case may be in your situation, the success of your project depends on being clear about your organization's specific needs and objectives.

As a group, you can agree on the criteria for success that allows you to determine whether an application can be feasibly migrated to Windows. Each application to be migrated to Windows should demonstrate feasibility (pass or fail) in four areas:

- **Feature/functionality**. The application must have the same features and functionality on the Windows platform as it does on UNIX. This test focuses on the primary features and functions, as defined by the users and their specific criteria.
- **Multitasking**. The applications identified need to operate in an environment conducive to user multitasking. During testing, additional applications should be launched to check of the ability of the application to share processor cycles.
- **Interoperability**. You may need to demonstrate that Windows users and UNIX users can coexist on the network, exchange files and otherwise collaborate on projects without the burden of a complicated file-conversion process.
- **Performance**. Compared to their speed on UNIX, applications must run as fast, if not faster, on Windows. You can measure performance using benchmarks where possible, or base tests on subjective analysis where no benchmarks are available.

Delivery of an agreed set of criteria marks the end of the initiation phase and the beginning of the project itself.

Cataloging and Managing Risks

Risk is the possibility of suffering loss. The loss could be in the form of diminished quality of the migrated application, increased development and/or support cost, missed deadlines or complete failure to achieve the mission and purpose of the project. In other words, a risk is a problem waiting to happen. It is recommended that the MSF approach to risk management be adopted, as illustrated in Figure 1. Risks, including potential showstoppers, can be identified and retired by identification, analysis, mitigation plans, tracking and control.

Figure 1. The MSF approach to risk management

This type of risk analysis and planning is often referred to as risk driven scheduling. This type of planning is of particular use in UNIX to Windows workstation migration, because of the overall scope

of a migration.

Identifying Risk

Throughout the assessment stage, you will have been spotting and documenting issues surrounding the migration. The issues may be business or technical, application or infrastructure specific. 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 should be done to minimize the probability of things going wrong.

In general, the more potential stumbling blocks—attitudinal, structural or otherwise—you identify ahead of time, the easier it is for you to minimize the amount of time spent trying to resolve them during testing.

Issues can be of a business, technical, application or infrastructure nature. You can now decide which issues are risks to the migration project. Note that a complete set of risks now allows for a thorough analysis later. You should make it clear that it is better to identify a risk and then determine it is of low priority at the analysis stage, rather than discounting it earlier on. This ensures more complete coverage, as well as reducing debate and demonstrating your willingness to listen to the project stakeholders.

Business risks

Effective risk management must consider the business environment in which the project operates. Many information technology projects fail, not for technology or project management reasons, but because of larger organizational pressures that are typically ignored.

Table 1 lists some examples of non-technical risk sources and possible consequences.

Table 1. Business risks

Categories of risk sources	Project consequences
Project resources	Cost overruns Schedule slips
Development process/environment	Inadequate functionality Poor product performance
Operational/Support environment	Demoralized staff Application instability through incorrect administration Insufficient physical space
End user	Customer dissatisfaction Lack of application support
Distribution of user base	Deployment problems Difficulties in training users

A major potential source of risk is the resistance to change by management, operations staff and users. UNIX is an established platform that enjoys a high degree of loyalty from many of its users, especially in terms of high-end applications, such as computer-aided design/computer-aided manufacturing (CAD/CAM), engineering and software development. Those advanced users who have developed custom functionality for their UNIX-based applications and tools through shell scripting may be especially fond of it. Thus, it is also important to determine whether some people in the organization might be biased against the project because of their devotion to UNIX. The following are questions that

should be addressed:

- How open is management to the idea of changing the environment?
- Is the value of the migration (that is, its benefits minus its costs) considered to be inadequate or negative? This question can be addressed at the risk analysis stage.
- Which groups might resist migrating to Windows? Groups could be organizational or geographical.
- Will users be open to the migration, open to it but with reservations, or totally opposed to it?

Additional risks may include unavailable personnel, and policy, procedural or legal issues. In addition to the risks, migration projects also have constraints such as:

- Hardware lease expirations
- Migration timetables
- Resources availability
- Expected cost savings

Failure to achieve any of the constraints could also be construed as a risk.

Technical risks

The application's technical context can be a major source of issues and risks. The following questions help you determine technical risks on the application migration:

- Is the way in which your systems and networks are administered going to present any difficulties?
- Does the required experience to perform the migration exist in-house, or is it available from outside?
- Do you have adequate technical support from the application vendors?
- Are there any specific constraints on standards conformance or security requirements—for example, is the application safety-critical?
- Would a delay to the migration cause any problems—for example, in terms of new technologies or versions of software being released?
- What contingency and disaster recovery plans are in place, and how will they be impacted by the migration?

Application risks

The application itself is a likely source of concern. You can identify potential risk areas by answering the following questions:

- Is the existing application code and scripts of sufficient quality to be ported? You should include configuration and installation scripts where these exist.
- How comprehensive is the existing development environment (for example, source control, configuration, build, debugging and bug tracking)?
- Is current application performance considered sufficient by its users, and how will this be impacted by the migration?
- Is there a shortfall between the existing application functionality and the needs of its users?
- Is the planned version of Windows too new for the UNIX application, or is the UNIX application simply incompatible with Windows?
- Is the technical complexity of the planned solution a cause for concern?

- Is backward compatibility required with the original application?

If you are porting a FORTRAN application, you should also consider the following risk areas:

- You will need a FORTRAN complier from a third party—is it compatible with other tools?
- Call level integration will likely be needed between FORTRAN and C/C++ code.
- A cross language build and debug strategy will be needed for Windows.

Infrastructure issues

The supporting infrastructure, and dependencies between infrastructure and application elements, can also yield issues and risks. The following questions should be addressed:

- Is the current hardware infrastructure suitable for running Windows, and/or is replacement hardware being sought?
- If the ported application is to exist in parallel with the UNIX application, will the two infrastructures coexist without difficulty?
- Can the network support the testing requirements of the project, particularly performance and scalability testing?
- Are there any dependencies with, or impacts on, other infrastructure elements, especially proprietary systems and applications, languages or scripts?
- Does the application depend on third-party software that is incompatible with Windows?
- Do you require to switch off any infrastructure elements before, during or after the deployment of the new application?

Additional risks may include configuration problems, driver incompatibilities, scheduling conflicts and resource conflicts.

Risk Analysis and Mitigation

The risk analysis stage involves taking each risk in turn and determining its constituent parts. You can consider a risk as being made up of:

- **Description**. The description must be in a language that the risk stakeholder (that is, who it impacts) understands.
- **Impact**. The impact is the effect that the risk would have, if it happened.
- **Probability**. The probability is the likelihood of the risk becoming real.
- **Severity**. The severity is the scale of impact that the risk would have, if it happened.
- **Cost**. It may be possible to assign a specific cost to each occurrence of the risk.
- **Mitigation**. Mitigation is the actions to be taken to ensure the probability of the risk is reduced or removed.

You can quantify risks in terms of both their probability and the severity that they would have. You should determine mitigation actions for each risk that is one or more actions to ensure that the risk is minimized or removed. Most risk factors can be mitigated with proper project management. However, there are some risks that can quickly escalate into showstoppers.

Note that you can also use the mitigation to quantify the risk. For example, if a mitigation action is easily implemented, it may be seen as worth doing even if the risk probability is low. Mitigations can be classed as hard, medium or easy, based on the perceived degree of difficulty to resolve the issue with the

stated solution.

Table 2 lists examples of issues that need resolution for the success of a migration project. Each issue is listed along with a proposed solution or mitigation strategy. The issue is then classified using a severity level defined as risk to the success of the project: high, medium or low—high severity means that the issue is a project showstopper if there is no resolution. Low priority means there are other workarounds or alternative solutions.

Table 2. Example issues

Issue	Impact	Mitigation	Severity	Implementation
Lack of support for X11R6. (Interix supports X11R5.)	Some of the application's X functionality could be compromised	Verify applications dependence on Release 6. Application may be able to use X11R5. X11R6 available from Interop Systems.	Low	Easy
Motif 1.2	Requires 3rd-party library	Use Motif 2.2 from Interop Systems.	High	Easy
Lack of support for xrt widgets from bluestone	Some of the application's X functionality could be compromised	Verify applications dependence on bluestone's xrt widgets vs. standard X toolkit. Need to verify availability of widget for Interix.	Medium	Medium
NIPC library needs to be ported to Interix subsystem	Could increase time/effort required to move the AT&T NIPC library to Windows/Interix.	Requires source code from AT&T, and assumes use of standard UNIX IPC mechanisms. Trail port to Interix to verify ported functionality.	High	Medium
Data complete source code port to Interix needs proof of concept trial port	Could increase time/effort required to move the MQSS application to Windows/Interix.	There were no dependencies discussed would make this a difficult port using Interix.	High	Easy

> **Note** FORTRAN migration risks are best mitigated by first defining the Windows development environment. This includes which FORTRAN compiler to use, and how to integrate with C/C++ code or third-party libraries. Second, modularity of the FORTRAN code combined with platform-specific feature extraction into a C/C++ compatibility layer also enhances the ability of the code to migrate from UNIX to Windows. This is also essential if the application needs to target both the UNIX and Windows platforms.

Managing the Risks

Risks should be included in a common repository and reviewed on a regular basis. You should publicize the top ten risks in the form of a risk assessment document. Not only does this ensure that everyone is aware of the risks, it also shows how you are proactively managing the migration project.

You can re-assess risks, and identify new ones, at regular intervals on the project. An opportunity to do this is at a milestone, where the reviews that take place should also incorporate a risk assessment. New risks should be included in the risk register as they are identified, whether or not a risk review is taking place.

However, note that there is no need for excessive risk management. Try to limit your risk assessment document to a maximum of ten high-priority risks most likely to threaten the project. You can then determine the additional contingency planning that is necessary at this point, taking into account the timeframe and objectives of the project.

Creating a Project Plan

The migration project plan describes the overall migration effort by gathering detailed plans, including development, test, training, deployment and end-user support plans, as well as the migration schedule. It should take into account the type of migration to be performed, the issues raised (and their mitigations), the funding provided and the availability and skill sets of the resources on the migration project.

The following sections describe each detailed plan and schedule.

Development Plan

The development plan describes how the development team will migrate the application. It describes information about the tools, methodologies, best practices, sequences of events and so on. It includes:

- Design goals (for example, components, services and technologies of the solution from the development team's perspective)
- Coding strategy (for example, rewrite vs. port, standards, tools, code re-use, libraries, things to avoid, portability, testability)
- Programming model, which:
  - Prescribes how the selected technologies will be used
  - Sets design guidelines as the design foundation for component specification; this helps to ensure consistency across the project
  - Uses different considerations to address different aspects of the solution
  - Provides a baseline for identifying potential technical risks
- Design and code review strategy (for example, comprehensive, formal review; casual, peer-based review; independent, third-party review)
- Source control strategy (for example, project file structure, version numbering convention, releases naming, installation programs/scripts, help files)
- Build strategy, which includes:
  - How (automated and/or manual, "make" structure), when, and in what frequency (should be performed on a regular and frequent basis)
  - The person or group responsible for the build
  - Test/feedback on the health of the build
  - Whether the build strategy will be linked with progress monitoring
  - The constraints on application builds
- Integration strategy; this includes how the different components will be integrated and how their

capabilities (functionalities) will evolve, for example:

- Identify an optimal order to integrate the different modules and components.
- Coordinate the integration order with the construction order so that modules and components are ready for integration at the right time.
- Define the integration strategy to facilitate the diagnosis of defects.
- Verify the components specification, to ensure smooth integration.

Test Plan

The test plan includes a testing strategy, the specific areas to be tested and the resources (hardware and people) the test team requires to do its job. The various categories of tests include:

- Coverage testing (used primarily during the development phase)
- Testing every feature of the product (for example, functional test)
- Testing the code base of the product (for example check-in, build verification and regression tests)
- Usage testing (used primarily during the stabilizing phase)
- Integration testing (used to verify the correct interactions between components and other systems). For integration testing, the test plan should cover:
  - Synchronization and coordination between components (for example, UNIX and Windows workstation data access by the migrated application)
  - Data integrity (especially when intensive file record-locking is required)
  - Features
  - Usage scenarios

For more information about application testing and the test plan, see Chapter 12, Testing and Quality Assurance.

Training Plan

The training plan includes a strategy and a plan for developing any necessary training materials for end users, administrators and support personnel. It includes a document that describes the training to be provided to members of the team to ensure that they have the capabilities to support the development and deployment process. The training plan defines training objectives and the audience; it also provides a training schedule that must be incorporated into the overall project schedule. The plan also describes the training vehicles, materials and resources.

Deployment Plan

The deployment plan includes a strategy and a detailed plan that can be used to prepare end users and operations personnel before and during deployment. It includes a document that describes all steps of the deployment process and details about the planning organization, and realization of:

- Installation strategy
- Deployment mechanisms
- Resources
- Contingency plans
- Site survey
- Systems support

Support Plan

The end-user support plan includes a strategy and details for developing a document that identifies the resources, processes and skills needed to ensure proper use of the system. It describes the different elements of user support provided, which might include one or many of the following types of support: such as:

- Reference manuals
- Online Help
- Online tutorials

There will be interdependencies between the support plan and the training plan, because some of the materials may be the same.

Migration Schedule

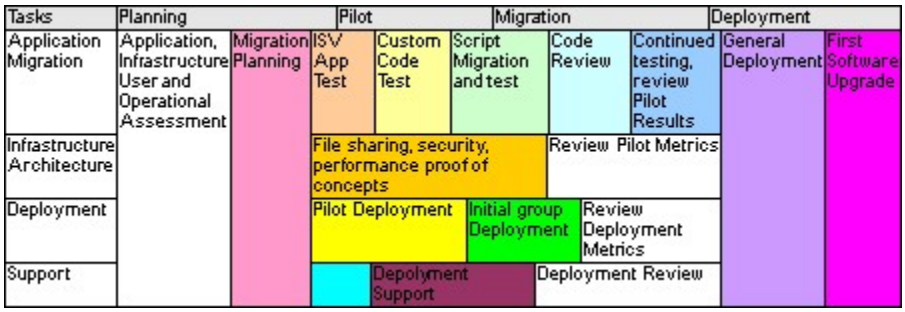
A migration schedule can have several components. These can be technical, business or organizational in nature. Common things to consider include:

- Windows versions
- Independent software vendor (ISV) release schedules
- Hardware lease expirations
- Hardware availability
- Resource availability
- Customer reorganizations

Keeping all of these dependencies in mind, visible project milestones need to be established. Clear metrics should be established for each milestone, with a post mortem review performed after each milestone.

Table 3 illustrates an example migration schedule.

Table 3. Example migration schedule



Each milestone should have a fixed date to keep the focus on the current task. A UNIX to Windows migration often needs a jumpstart to get moving. You should pay extra attention in the early stages until the migration gains the momentum to see it through to the finish.

Planning the Project Resources

It serves little purpose to have the most comprehensive and well-considered migration plan in the world if you don't have the time and resources to implement it. It is unlikely that infinite resources are

available, so migration activities are bounded by available effort and financial support. Understanding the overall objectives, being aware of potential benefits and being clear about what you must produce as a result of the migration project gives you the foundation you need to determine the resources and skill sets you need to complete the project successfully in the time allotted.

The resources and funds available to the project may already have been fixed by the project sponsor. If not, you should determine available physical and human resources at this stage so that you are aware of the resources and funds you have to begin with.

Sizing the Effort

During the assessment and analysis phase, you will have produced a business case that determined (in broad terms) the amount of effort required for the various options available to you. You can now decide whether to conduct a more detailed sizing study or to press ahead with the project based on the broad figures.

Migration projects are notoriously difficult to size, even with the most in-depth study. An appropriate strategy is to consider the migration as a series of deliveries, each corresponding to a subset of the requirements. The most essential requirements (and those on which they depend) should be part of the first delivery, with subsequent deliveries corresponding to less urgent requirements. This ensures that, if the project runs out of time or money, at least the significant requirements are dealt with. In addition, it may be possible to deploy the essentials of the migrated application early, so that its users can derive early benefits. Even if they cannot use the full application, the core migrated application can be used for:

- User training
- Early validation of user requirements
- Scalability and performance testing

Time is one of the most important factors in determining the scope and quality of any project. Among the questions you need to consider are:

- Do you have (for example) three months, three weeks or three days to get the job done?
- How much time can you apportion to planning?
- How much time must you dedicate to testing?
- How much time can you set aside to deal with unforeseen issues?
- How long will it take to produce a final deliverable?
- What additional resources may be available from outside the project; for example, could users be deployed as testers?

Picking the Team

Early in the project cycle, the people who will make up the overall project team must be chosen. This obligation is somewhat separate from the process phases of envisioning, planning, development and testing. Nevertheless, the team members should be chosen in the envisioning phase or early in the planning phase so that their input can be beneficial to the overall process.

Whether the team is large or small, the MSF team model prescribes six goals for success. Every team must strive for:

- Satisfied customers

- Delivery within project constraints
- Delivery to specifications that are based on user requirements
- Release only after addressing all known issues
- Enhanced user performance
- Smooth deployment and ongoing management

The overall project team is divided into functional teams that interact with each other as peers, not in a hierarchical management-style structure. Each functional team and each team member has a well-defined role and a specific mission on the project. The functional teams are product management, program management, development, testing, user education and logistics management.

The roles of the team model correspond directly to the six quality goals for an effective project team. Each team is responsible for assuring that one of the goals, the team goal, is completed, as follows:

- **Product management** has the goal of assuring satisfied customers. Customers of the migration project include the sponsor who is paying for the migration and the end users of the application being ported.
- **Program management** is charged with delivery within project constraints. To meet this goal, program management owns and drives the schedule, the features and the budget for the project. As discussed earlier, there may be multiple, interdependent migration projects; and there will certainly be multiple deliveries within each project.
- **Development** must deliver the project to meet the product specification, which is based on customer requirements. The developers are the team members responsible for porting the application itself.
- **Testing** assures that the project releases only after addressing all issues. A migration issue is anything that prevents the migrated application from meeting its requirements. The testing team owns the testing process, test strategies, test plans and the testing schedule.
- **User education** is charged with producing enhanced user performance so that users have increased productivity from using the product. There are specific pressures on user education from the migration project, especially because of the potential for resistance to change.
- **Logistics management** focuses on smooth deployment and ongoing management. It acts as the advocate for operations, product support, help desk and other channels linked to the deployment of the migrated application.

Depending on the size of the project, team and individual roles may be shared by more than one person on a team, may be given to an individual, or one person may have several roles. For larger projects, each role may be assigned to a single person or a team of people with a team lead. For very small projects, a small number of people may take several roles within the framework.

For further information, see the Microsoft Solutions Framework.

Subject Matter Experts

Having the right people involved with the project from the first day is a key factor for its ultimate success. The people that need to be involved include:

- A sponsor who will champion the project with management and the customer
- Technical experts from each area of the application including source, the build procedures and scripting
- An information technology infrastructure expert who understands security, connectivity and

installation issues
- A testing expert who understands the functionality of the application
- Developers who are experienced in either or both UNIX and Windows platforms

Having a sponsor from upper management helps promote the project to management and the customer. Because the sponsor is one of their own, the corporate decision makers are much more receptive to input from that person. The sponsor is also in a position to help if roadblocks or problems arise that can be solved only at the executive level. The product management role should be taken by the sponsor, if possible.

Technical experts with UNIX knowledge are responsible for analyzing the application sources, makefiles and scripts to determine their cross-platform compatibility. Along with experts from the Windows side, they are also instrumental in making any modifications necessary to get the code running on the Windows platform. The role of the technical experts from both disciplines is to make architecture decisions and to implement required changes. The technical experts fill the development role and may be in other roles.

An expert from information technology assures that infrastructure issues do not derail the migration effort. This person must know about security within the organization and the steps that are necessary to give the team required access rights. The developers need to be able to get sources and run tools on both platforms. The end users will need to run the finished product and have access to any shared resources such as databases. The information technology expert may also be required at the beginning of the project to deploy application development and productivity tools. The information technology expert fills the logistics management role.

A testing expert allows the team to meet the goal of releasing only after addressing all issues. The testing expert formulates the test plan for the migrated application. Because developers make notoriously bad testers, a person not on the development team must fill the testing role.

The ideal skill set for a developer working on the migration project is a mixed background in both UNIX and Windows development. The primary software development skills required are:

- Development languages (for example, C, C++ and/or FORTRAN)
- Build and runtime environments (for example, **make**, **gmake**, **ksh**, **csh**)
- UNIX skills covering the architecture being ported (for example, X/Motif GUI, IPC-based services)
- Win32 development skills (if very much of the code will be redesigned/rewritten)

Other important skills include a background in porting and other specific skills, depending on the nature of the application (for example, GUI or client-server development skills).

Training the Team

Team members that don't have all of the required skills should get training before the project implementation starts. This type of concern is usually in the areas of development and/or logistics management. The exact training required depends on the nature of the migration project; for example, if the migration will be to port an existing application, then the appropriate training would be for the development environments, such as Microsoft Visual Studio® development system or Microsoft Visual C++® development system. Training in the Windows application programming interfaces (APIs) would also be helpful. Similarly, if you plan to use third-party tools either to aid in the port or as development

libraries, then training from the vendor may be required.

If the migration from UNIX to Windows is using the Interix subsystem, then the team needs to understand the capabilities of Interix. The key here is to understand the differences between Interix and the version of UNIX that currently hosts the application. Because Interix is not UNIX, there is a chance that some issues will arise.

Training is available from Microsoft Certified Technical Education Centers (Microsoft CTECs) in many cities around the world. These centers offer instructor-led courses to teach you about all aspects of the Windows platform. Some useful material is also available online in Web format and in self-paced book or CD format from Microsoft Press.

Planning Execution

To ensure a successful migration, a number of techniques should be carried out throughout the migration project. These include:

- Milestones and checkpoints, to ensure that the project plan is followed and to reveal any risks as they arise
- Frequent delivery, to enable quality checks and to exercise the migrated software
- Metrics, to provide visibility on project progress

At the end of the project, you should conduct a closedown review to ensure that any valuable lessons that have been learned are documented and to check off any completion actions, such as final sign-off of deliverables and archiving of documents and earlier software versions.

Milestones and Checkpoints

The migration process should be based on milestones and checkpoints or review points. Milestones are a definite, scheduled point in time when a material object must be delivered. Frequent checkpoints verify that the team is making progress toward the goal of delivering the completed software.

The shorter the project, the more often you should have a checkpoint or review to assess progress. For example, during a two-week project, there should be short, daily reviews at the very least. The sooner you correct a problem, the easier it is to fix and the more time you have to devote to delivering the full functionality on time. If this seems extreme, think about a year-long project, which would typically call for weekly status meetings. That would be fifty-two meetings over the course of the project. For a two-week project, daily meetings would amount to a total of only ten. Percentage-wise, the project is allowed to go much farther between checkpoints.

Frequent Delivery

Each stage of the project should have one or more concrete deliverables, each of which is appropriate for that stage of the migration. The milestone for each phase is the completion of the deliverable item. For example:

- **Assessment and analysis**. The deliverable of this phase is a business case that is supported by a comprehensive set of analysis results.
- **Planning**. The deliverable of this phase is a functional specification, migration plan and risk catalog.

- **Development environment**. The deliverable of this phase is a working environment, with code imported and ready to migrate.
- **Development**. The deliverable of this phase is the completed code, which compiles to result in a working, testable application.
- **Testing**. The deliverable of this phase is an application that conforms to its specification with all appropriate bugs addressed.
- **Deployment**. The deliverable of this phase is a statement of user satisfaction.

Larger migration projects have multiple deliveries from the development phase. Product release is the final project milestone that culminates the testing and deployment phase. This means that the project is complete and that the vision has been achieved. The next phase, closing down the migration project, is actually the beginning of the next loop in the life cycle of projects.

Two additional techniques used by Microsoft are doing daily builds and using the software being developed which is known as "eating your own dog food." Doing daily builds of the entire code base ensures that no problems have been introduced that prevent the whole system from begin created. Eating your own dog food means that the developers exercise the new software every day. Clearly these techniques may be difficult to apply early in the migration process, which relies on the migration of the build environment. However, the number of builds and the use of the resulting application should increase throughout the migration.

Project Metrics

Having the team define and track metrics is another procedural item that can help with tracking progress. It is common to measure items such as function points, requirements achieved, bug counts or number of lines of code written or converted. The development and testing teams should pick a set of metrics that are useful in tracking migration progress. To be useful to the whole team, these metrics should be made available on a regular basis. Whether this is done through e-mail messages, a team intranet Web site, a share on a server or by other means, is up to team members. Whatever the choice, it should fit into their normal routine so that the process of checking the figures is a natural extension of their work habits.

Do users expect a summary document, a detailed report of the test cases or a recapitulation of the testing, test results and migration strategy? Further, do they expect any progress reports as you carry out your testing? Knowing what you need to produce from the testing helps you to focus and structure your approach to the testing itself.

Closing Down the Migration Project

At the completion of the migration project, the project team should analyze the results and produce documentation that can be used in subsequent migrations. The documentation may include a paper that describes what was done and what happened. The paper should include details of which things worked, but more importantly, it should include information about what did not work as expected and what was done to fix the problems. For example, the report should include information about significant porting problems such as:

- Availability of required tools (compilers, debuggers, analysis)
- Lack of third-party libraries (math, X Windows)
- Infrastructure issues, including file access security
- Makefile issues

- Script usability
- API migration issues

In addition to creating documentation, program management may want to create presentations for the customer or upper management.

As the final action to conclude the project, the team should decide how to proceed in the future. The next step is often the beginning of another project with the team envisioning the migration of other, larger applications. If the migration project runs into significant issues, the team should address these through use of prototyping before beginning a more complex migration. With the prototype project completed, the organization can begin moving significant, line of business applications to the Windows platform. By following the framework for software development, the organization has valuable experience in both the tools required and the process for migrating from UNIX to Windows.