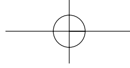
Mendoza_01.qxp 3/22/2006 8:51 AM Page 1



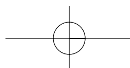
# Chapter 1

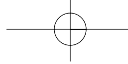
# Porting Project Considerations

## In this chapter


- 1.1 Software Application Business Process page 2
- 1.2 The Porting Process page 3
- 1.3 Defining Project Scope and Objectives page 8
- 1.4 Estimating page 10
- 1.5 Creating a Porting Project Schedule page 16
- 1.6 Porting Process from a Business Perspective page 17
- 1.7 Annotated Sample Technical Questionnaire page 18
- 1.8 Summary page 26







Application porting refers to the process of taking a software application that runs on one operating system and hardware architecture, recompiling (making changes as necessary), and enabling it to run on another operating system and hardware architecture. In some cases, porting software from one platform to another may be as straightforward as recompiling and running verification tests on the ported application. In other cases, however, it is not as straightforward. Some large applications that were written for older versions of operating systems and compiled with native compilers may not adhere to present language standards and will be more difficult to port, requiring the full attention of project management.

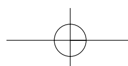
This chapter supplements currently available project management materials and books about application porting projects. Topics such as how to use formalized requirements processes, how to better communicate between software developers, and how to practice extreme project management are topics that modern-day project managers are already well informed of. However, because software development is not exactly the same as porting and migrating software, a gap exists in current publications—a gap that this chapter addresses.

The focus here is on software porting and migration process details and technical risks. All information in this chapter has been collected and gathered through countless porting engagements. This chapter presents best practices and methodologies to achieve high-quality ported applications. This chapter specifically appeals to those involved in software porting and migration projects, especially project managers and application architects.

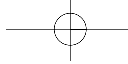
## 1.1 Software Application Business Process

Before embarking on a porting project, it is important to note which business processes within the application life cycle will be affected. Business processes currently in place must be modified to accommodate the ported application in terms of having a new platform to support, new test environment, new tools, new documents, and (most important) new customers to support and establish relations with.

The three main areas of the application life cycle to consider that may affect the business process are development and testing, customer support, and software application distribution:



Mendoza_01.qxp 3/22/2006 8:51 AM Page 3





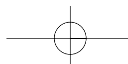
- **Development and testing.** Within the development and testing organization, personnel must be trained to develop and test on Linux in terms of application programming interface (API) differences, development tools, debugging tools, performance tools, third-party applications needed by the application to be ported, and performance tools.
- **Customer support.** Within the customer support organization, support personnel must be trained on Linux system administration, third-party applications needed by the ported application, installation and administration procedures, package maintenance options in the Linux environment, debugging tools and procedures, and anything else that needs to be learned to provide customer support for the ported application.
- **Software application distribution.** Within the software application distribution organization, sales and consultants must be trained in overall Linux features and capabilities.[1] Personnel in the software distribution channels must be trained as trainers of the software application in a Linux environment. From a customer's perspective, they will also look for integration skills to integrate Linux within their IT infrastructure.

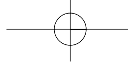
Porting an application to Linux means modifying business organization processes that can be affected by a newly ported application. The three main areas should be considered and included in the overall project plan before starting the porting process.

## 1.2 The Porting Process

Developers involved in porting projects tend to follow similar steps when porting any software application. These steps include scoping, analyzing, porting, and testing. Each step in the process builds a foundation for the next step in the process. Each step, when done properly, makes the next step of the process easier to accomplish. Figure 1-1 shows the high-level steps taken during a porting project.

1 Different Linux distributions exist that may require separate training sessions for each.







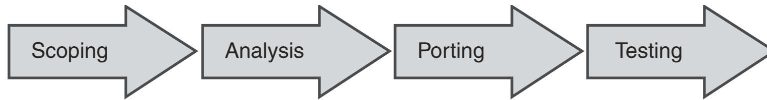



FIGURE 1-1
High-level steps taken during a porting project

### 1.2.1 Scoping

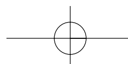
Scoping is the step in which the project manager asks the porting expert[2] and a domain expert[3] to get together to identify the products, development, and test environment the application to be ported relies on. The key areas to identify during the scoping process include product dependencies, development environment components, build environment components, and test environment components:

- **Product/software dependencies.** Identifying which products the application to be ported relies on means determining which versions of database, middleware, and third-party libraries it uses. By knowing the products and versions, the porting expert can assess whether those products and versions are available for the Linux platform.
- **Development environment components.** Assessing the development environment includes identifying what programming language the application is written in. Applications written in a more recent programming language such as Java tend to port more easily, whereas applications written in C or C++ usually require more analysis and porting effort.
- **Build environment components.** Assessing the build environment includes making sure the build tools are available on Linux. Platform-dependent compiler and link flags used on the source platform must be investigated to determine whether equivalent flags exist on Linux. Some build environments may have dependencies on the source platform that will require a little more effort to port to Linux.

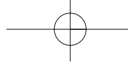
---

2 A software developer who is experienced in porting applications and has knowledge of source and target platforms as well as other third-party products the application uses. We also refer to this person as a porting engineer in this book.

3 A person who is knowledgeable about the application to be ported. This could be the application architect or the chief developer of the application.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 5





- **Test environment components.** Identifying the test environment for the ported application leads to questions pertaining to the ownership of testing after the application is ported. Usually porting engineers do unit testing for the part of the application they port and then hand it off to a testing group for more verification and systems tests. But who is the testing group?

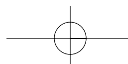
In most cases, the scoping step leads to identifying associated risks that will be assumed by the project as a whole when it begins. Some risks that can be identified in the scoping step are included in the following scenarios:

- Versions of the database, middleware, or third-party libraries are not available for the Linux platform.
- The application has some assembler routines that need to be translated to assembly language instructions on the Linux platform.
- The application was written using a set of APIs or a programming model unique to the source platform. This also includes assumptions about such things as word size or “endian-ness.”
- The application was written to draft-standard C++ semantics that relied on the source platform’s native compiler framework.
- The test environment requires a complicated client/server framework.
- The development environment requires third-party tools that need to be compiled or ported to the Linux platform.
- The application’s distribution or installation method requires facilities or tools unique to the source platform.

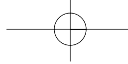
Scoping is an involved step in the porting process that takes into account every new piece of information that can be learned from asking the right questions—questions about documentation, packaging, and performance tuning, to name a few. These and others are mentioned in the “sample porting questionnaire” near the end of this chapter.

### 1.2.2 Analysis

There are two views in the analysis step of the porting process: a project management point of view, and a porting point of view. From a project management point of view, analysis is the step that assesses the various porting issues and risks



Mendoza_01.qxp 3/22/2006 8:51 AM Page 6





identified in the preceding step and what impact they bring to the porting project as a whole. The analysis step involves the formulation of the project plan, which includes identifying scope and objectives, creating work schedules, procuring resources needed, and assigning roles within the project.

Identification of scope and objectives defines the boundaries and responsibilities of the project manager and the members of the project team. Boundaries are clearly identified sets of work that need to be done for the project. For example, a simple statement such as "Module A in application XYZ needs to be ported and tested on platform B" can be a good start to defining the boundaries of the work.

After boundaries have been identified, tasks for the porting work can be defined, leading to a work breakdown schedule.[4] The work breakdown schedule helps define which tasks need to be done and whether those tasks can be done in sequence or in parallel. In addition, the work breakdown schedule identifies the resources needed. An overall schedule for the entire porting project results from identifying tasks and resources needed.

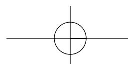
From a porting point of view, analysis is a step in the porting process during which a porting engineer examines the application architecture in more detail. The porting engineer begins to identify the APIs and system calls used in the application. In addition, the application is assessed as to its use of dynamic linking and loading, networking, threads, and more. This analysis feeds information back to the project manager that can be used to formulate more detailed tasks and accurate schedules.

### 1.2.3 Porting

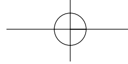
Porting is the step in the process during which the porting engineers perform their assigned tasks. Depending on the application and the work breakdown schedule that resulted from the preceding step, porting engineers may only be able to work their assigned tasks in a serial fashion. Serial tasks are a product of tightly coupled applications. This means that some modules of the application are highly dependent on other parts of the application and that these modules can be worked on only when the part they depend on is ported. A prime example of this is the build

---

4 A project management term used to denote work as an activity arranged in a hierarchical order that has tangible results otherwise referred to as a deliverable.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 7





environment. If the build environment was designed to build the whole application as a monolithic procedure, chances are all modules depend on common configuration files that need to be modified before any porting work can be performed.

If the porting tasks are independent of each other, however, the work effort can be parallelized. Loosely coupled modules can be worked on separately and simultaneously by different porting engineers. A key example of this are shared libraries that can be built independently of each other, do not have common components, and exist for other modules to build on. Identifying which tasks can be done in parallel is important and should be done as part of the analysis step of the process.

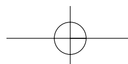
The porting engineers' task of compiling the code on the Linux platform includes identifying and removing architectural dependencies and nonstandard practices if possible. Identifying and removing architectural dependencies means heeding compiler errors and warnings produced at compile time and correcting them as needed. Removing architectural dependencies and nonstandard practices involves examining and changing the code to use more portable structures or coding standards. Experienced and quality-conscious porting engineers usually do the latter as they correct compiler errors and warnings.

The porting effort includes porting the build environment to the Linux platform, too. This task should be identified during the scoping step. Although some build environments are portable, some are not. Making sure the build environment does not create any potential problems is a task easily overlooked and needs to be scoped and analyzed with extreme caution.

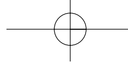
After the application has been ported—meaning it compiles on the Linux platform without errors—the porting engineer is expected to run unit tests on the application. The unit tests can be as simple as executing the application to determine whether it produces any runtime problems. If runtime problems are detected, they are debugged and fixed before the application is handed to the testing group; the test group then runs more thorough tests on the ported software.

### 1.2.4 Testing

During testing, the assigned testers run the ported application against a set of test cases, which vary from simple execution of the application to stress-type tests that ensure the application is robust enough when run on the Linux platform. Stress testing the application on the target platform is where most porting problems



Mendoza_01.qxp 3/22/2006 8:51 AM Page 8





related to architectural dependencies and bad coding practices are uncovered. Most applications—especially multithreaded applications—behave differently during stress testing on a different platform, in part because of different operating system implementations, especially in the threading area. If problems are found during testing, porting engineers are called in to debug and fix them.

Some application porting may also include porting a test harness to test the application. Porting this test harness is a task that is also identified during scoping and analysis. Most of the time, the testers need to be trained on the application before they can test it. Learning about the application is a task that is independent of the porting tasks and can be performed in parallel.

If bugs are found, they are fixed and the source recompiled; the testing process repeats until the application passes all testing requirements.

### 1.2.5 Support

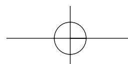
When the port is complete, the development phase ends, and the support phase begins. Some of the porting engineers are retained to help answer any porting-specific questions the customer may have. In addition, the developers are expected to train the customer on configuring and running the application on the Linux platform. The typical support phase lasts from 60 to 90 days after the port. During this time, the porting engineers train and answer technical support and sales personnel questions regarding the newly ported application on the Linux environment. After the technical support and sales personnel have been trained in the newly ported product, the porting engineer's task is done.

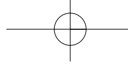
## 1.3 Defining Project Scope and Objectives

Project scope is defined as the specific endpoints or boundaries of the project as well as the responsibilities of the project manager and team members.[5] A clearly defined project scope ensures all stakeholders (individuals and organizations that are involved in or that might be affected by project activities—project team, application architect, testing, customer or sponsor, contracts, finance, procurement, contracts and legal) share a common view of what is included as the objectives of the project.

---

5 *Radical Project Management*, Rob Thomsett, 2002, Prentice Hall







Clear project objectives/requirements lead to a better understanding of the scope of the project. During the analysis step in the porting process, customer objectives and requirements are gathered, and these objectives transform themselves into work breakdown structures and eventually project deliverables. Project objectives and requirements provide a starting point for defining the scope of the project. After all project objectives have been laid out, the project scope becomes more defined.

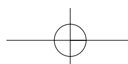
One way to define the scope of a project is to list objectives that will and will *not* be included in the project. A requirements list from the customer is a good start. After the requirements have been gathered, the project manager and the technical leader review the list in detail. If there are questions about the requirements list, they should appear in the "not to be included" list of objectives, at least initially. The list is then reviewed by the customer, who may make corrections to the list. The list is revised until all parties agree that it presents the true objectives of the project.

Again, it should be sufficiently detailed so that it lists requirements that will and will not be included in the project. Yes, it is really that important to include details that are beyond the scope of the project. Objectives that do not provide enough definition to the scope of the project may turn out to be a point of contention as the project draws closer to its release date. An example of this might be listing how pieces of the porting work will be delivered to the porting team—will it be in phases, or will it be in a single delivery? Will each delivery be considered a phase, or will there be smaller pieces of work within a phase? How many phases will make up the entire project? The list not only defines the project itself, it also sets the expectations for all stakeholders in the project.

Here are the basic rules to follow when creating the project objectives list:

1. Define project scopes in as much detail as possible.
2. Confirm that all stakeholders agree to project scopes.
3. List unresolved items on the "not-included/not-in-scope" list until they are resolved.

Table 1-1 shows an example of a simple "in-scope" and "not-in-scope" table for a sample porting project.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 10

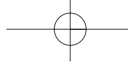




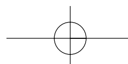
TABLE 1-1
In-Scope and Not-in-Scope Objectives

| Project: Customer *X* Application Port to Linux | |
|---|---|
| **In Scope** | **Not in Scope** |
| Port Directory, Mail, and Order Entry modules to Linux to be divided up in three phases. Each module will be one phase.<br>Functional verification tests on these modules will be conducted by porting team. Definition of functional verification tests and acceptance criteria will be defined in another section.<br>Provide debugging assistance to customer developers during system verification tests. | System tests of in-scope modules are not included in the porting project. Customer will be responsible for systems test.<br>Packaging of application is not in scope.<br>Fixing bugs found to be present in the original source code are not in scope.<br>External customer documentation of ported modules is not in scope. Customer to produce external customer documentations. |

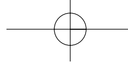
Part of defining the objectives is listing the acceptance or success criteria. A consensus regarding the porting project success criteria must be reached by all stakeholders. Project success may mean passing a percentage of system tests on the Linux platform or passing a level of performance criteria set out by the systems performance group—that is, if the project objectives are to run a specific set of tests or performance analysis, respectively. Regardless of how the project success is defined, all stakeholders must understand and agree on the criteria before the porting effort starts, if possible. Any changes to the criteria during the course of the porting cycle must be communicated to all stakeholders and approved before replacing the existing criteria.

## 1.4 Estimating

Many porting projects go over budget and miss schedules because risks were not managed properly. Risks play a major part in estimating the schedule as well as resources needed for porting projects. In an application porting project, such risks come from different aspects that relate to application porting, including the following:



Mendoza_01.qxp 3/22/2006 8:51 AM Page 11





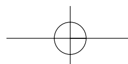
- Skill levels and porting experience
- Compiler used
- Programming language used
- Third-party and middleware product availability
- Build environment and tools
- Platform-dependent constructs
- Platform- and hardware-dependent code
- Test environment setup needed
- User interface requirements

Depending on the application to be ported, each of these aspects presents varying levels of complexity and risks to the project. Assessing complexity and risk level help you determine whether they are manageable.

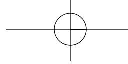
### 1.4.1 Skill Levels and Porting Experience

The most glaring difference between porting applications and software development is the skill set of the programmers. Although software application developers tend to be more specialized in their knowledge domain, software developers who do porting and migration need broader and more generalized skill sets. An application developer can be someone who is an expert in Java and works on a Windows development environment. Another may be a programmer specializing in database programming on a Sun Solaris operating system environment. On the other hand, engineers who port code are expected to be experts in two or more operating system platforms, programming languages, compilers, debuggers, databases, middleware, and the latest Web-based technologies. They are expected to know how to install and configure third-party database applications and middleware.

Whereas application developers tend to be specialists, porting engineers need to be generalists. Application developers may work on an application for about 18 months (the typical development cycle), whereas porting engineers work on a 3- to 6-month project cycle and are ready to start on a new port as soon as the last one is finished. Finding skilled porting engineers who fit the exact requirements of a porting project may be difficult at times. This is most true when the porting efforts require porting from legacy technologies to newer ones.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 12





### 1.4.2 Compiler

The compiler and compiler framework used by the source platform make a big difference when porting applications to the Linux environment. If the same compiler is used in both the source and target platforms, the task becomes easier. An example in this case is when the GNU compiler is used in both the source and the target platform. Besides the `-g` and `-c` flags, different brands of compilers use flags differently. Compiler differences become more difficult if the programming language used is C++.

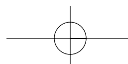
Another thing to consider is the version of the compilers. Source code compiled a few years back with older versions of compilers may have syntax that does not conform to present syntax checking by compilers (because of standards conformity). This makes it more difficult to port on different compilers because these compilers may or may not support backward compatibility for standards. Even if the compilers support older standards, different compilers may implement support for these standards differently.

### 1.4.3 Third-Party and Middleware Product Availability

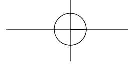
The complexity of the port increases when the application to be ported uses third-party and middleware products. Complexity increases even more when the versions of those products are not available for the target platform. In rare cases, some third-party products may not even be available on the target platform. In the past few years, middleware vendors such as IBM, Oracle, Sybase, and many more have ported their middleware products to Linux. They have made efforts to make their products available for companies that are ready and willing to make Linux their enterprise platform. This is partly why we are seeing more and more companies willing to port their applications to Linux.

### 1.4.4 Build Environment and Tools

The simpler the build environment, the less time the porting team takes to understand how to build the source code. More complicated build environments include multipass builds where objects are built with different compiler flags to get around dependencies between modules. After a first-pass build to build some modules, a second pass compiles yet more modules, but this time builds on top of the previously compiled modules (and so on until the build completes). Sometimes the build scripts call other scripts that automatically generate files on-the-fly based on



Mendoza_01.qxp 3/22/2006 8:51 AM Page 13





nonstandard configuration files. Most of these files may be interpreted as ordinary files to be ported, but in fact, it is really the tools and the configuration files that need to be ported to work on the target module. These types of tools are mostly missed during assessment and analysis, which can result in less-than-perfect schedules.

One build environment that came into fashion sometime in the early 1990s was the use of imake. Imake is a makefile generator intended to ease build environment portability issues. Instead of writing makefiles, one writes imakefiles. Imakefiles are files that contain machine-independent descriptions of the application build environment. Imake creates makefiles by reading the imakefile and combining it with machine-dependent configuration files on the target platform. The whole concept of architecting a build environment around the imake facility to make it portable is a noble idea. Of the most recent 50 porting projects done by our porting group, only a few were built around the imake facility. Unfortunately, all of them needed as many modifications to their own environment and imakefiles, negating the ease that imake is intended to provide. And because imake was rarely used, anyone who needed to do a port needed to "relearn" the utility every time.

Nowadays, makefiles are written for the open source make facility called GNU Make (or gmake for short). Most makefile syntax used in different UNIX platforms is syntax that gmake can understand or has an easy equivalent to. Architecting applications around the gmake[6] facility is the most accepted and widely used method in build environments today.

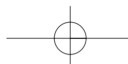
Source code control is another aspect that must be considered. In the Linux environment, CVS is the most frequently used source code control environment. Other source code control environments are Subversion[7] and Arch.[8] A porting project needs to have source code control if there is a possibility of several porting engineers working on the same modules at the same time.

---

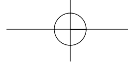
6 Some may favor Autoconf even though it had a certain level of complexity compared to using plain old makefiles.

7 http://subversion.tigris.org

8 www.gnu.org/software/gnu-arch/



Mendoza_01.qxp 3/22/2006 8:51 AM Page 14





### 1.4.5 Platform-Dependent Constructs

When porting from UNIX platforms based on RISC architecture to x86-based platforms, a chance exists that application code needs to be assessed for byte-endian dependencies. For example, the application implements functions that use byte swapping for calculations or data manipulation purposes. Porting the code that uses byte-swapping logic to an Intel-based machine, which is little-endian architecture, requires that code be modified to adhere to little-endian semantics. In cases where byte-swapping logic is not changed between platforms, debugging becomes a chore because it is difficult to track where data corruption takes place. When this happens several instructions before the application fail. Make sure to identify platform-dependent constructs in the scoping and analysis steps.

### 1.4.6 Platform- and Hardware-Dependent Code

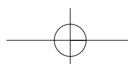
Applications that require kernel extensions and device drivers to operate are the hardest to port. Kernel APIs for all platforms do not follow any standards. In this case, API calls, number of arguments, and even how to load these extensions into the kernel for a specific platform will be different. In most cases, new code that needs to run on Linux must be developed. One thing is for certain: Kernel code will usually, if not always, be written in C or platform-dependent assembler language.

### 1.4.7 Test Environment and Setup

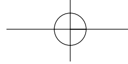
When the port is done and the project scope includes system tests and verification, equipment required to test the code may contribute to the port's complexity. If the application needs to be tested on specialized devices that need to be ordered or leased, this can add complexity to the project. If the application needs to be tested in a complicated clustered and networked environment, setting up the environment and scheduling the resource adds time to the project.

Testing may also include performance testing. Normally, performance tests require the maximum configuration setup one can afford to be able to load up the application to see how it scales in big installations.

Porting of the application test harness needs to be included in the porting schedule. A test harness that includes both software tests and specialized hardware configurations needs to be assessed as to how much time it will take to port and configure.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 15





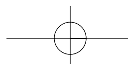
### 1.4.8 User Interface Requirements

Porting user interfaces can range from being easy to complex. User interfaces based on Java technology are easy to port, whereas user interfaces based on X11 technology are more complex. In this type of application, the code may be written in C or C++.

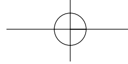
Table 1-2 is a summary of the various aspects involved. Each aspect is ranked as easy, medium, or high complexity, depending on the technology used for each aspect.

TABLE 1-2
Software Porting Risks and Complexities

| Aspect | Easy | Medium Complexity | High Complexity |
|---|---|---|---|
| Compiler/ programming language | Java<br>Shell scripts | Use of C, COBOL, Fortran language<br>Use of nonportable syntax<br>Original source code written to other standards of compiler | C++ using compiler-specific framework that supports varying levels of the C++ standard<br>Source code written in languages other than Java or C<br>Use of nonstandard compilers such as ASN or customer-built interpreters<br>Use of assembler language<br>Use of RPG language |
| Use of third-party products or middleware | None | Supported and available on target platform | Not supported on target platform<br>Use of third-party tools written in C++ |
| Build environment and tools | Simple makefiles | Combination of makefiles and build scripts | Use of noncommon build tools such as imake, multiple-pass builds, on-the-fly-generated code modules |
| Platform/ hardware-dependent code | No platform- or hardware-dependent code | Platform/hardware-dependent code comes from third-party products already ported to target platform | Extensive use of kernel extensions and device driver code |
| Test environment and setup | Stand-alone | Client/server setup | Networked, high availability, clustered<br>Needs external peripherals to test such as printers, Fibre Channel-attached disks |
| User interface | Java-based | Standard-based interface such as X11, CDE | Nonportable user interface such as custom interfaces using proprietary code |



Mendoza_01.qxp 3/22/2006 8:51 AM Page 16





Experience shows that real efforts needed to port the application are uncovered during the first two or three weeks of the porting project. This is the time when the code is delivered to the project team and porting engineers get a first glimpse of the software application. In other cases, this is the first time porting engineers work on the application in a Linux environment. They begin to dissect the application components and start preliminary build environment setups. Within the first two to three weeks, the porting engineers will have configured the build environment and started compiling some modules using GNU-based tools and compilers.

After two to three weeks, it is good to revisit the estimated time schedules to determine whether they need to be adjusted based on preliminary real experiences of porting the application into the Linux platform.

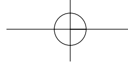
## 1.5 Creating a Porting Project Schedule

When creating the porting schedule, consider all risks, including technical and business-related issues. In terms of technical issues, common things such as resource availability, hardware availability, third-party support, and Linux experience need to be considered. On the business side, issues such as reorganizations, relocations (moving offices), customer release dates, and business objective changes may affect the porting schedule as a whole.

These technical and business issues will create several dependencies external to the porting project that may not be controlled; therefore, it is advisable to consider each external issue to mitigate risks accordingly.

Creating schedules for porting projects is just like creating schedules for software development projects. Figure 1-2 is an adaptation of a diagram from *Radical Project Management* by Rob Thomsett (Prentice Hall, 2002, p. 191). The figure shows that the average estimation error made before the project scopes and objectives are cleared up is +400 percent and –400 percent. As the objectives, scope, risks, and other technical and business issues are clarified, the estimated porting schedule comes closer to the real schedule.

Mendoza_01.qxp 3/22/2006 8:51 AM Page 17





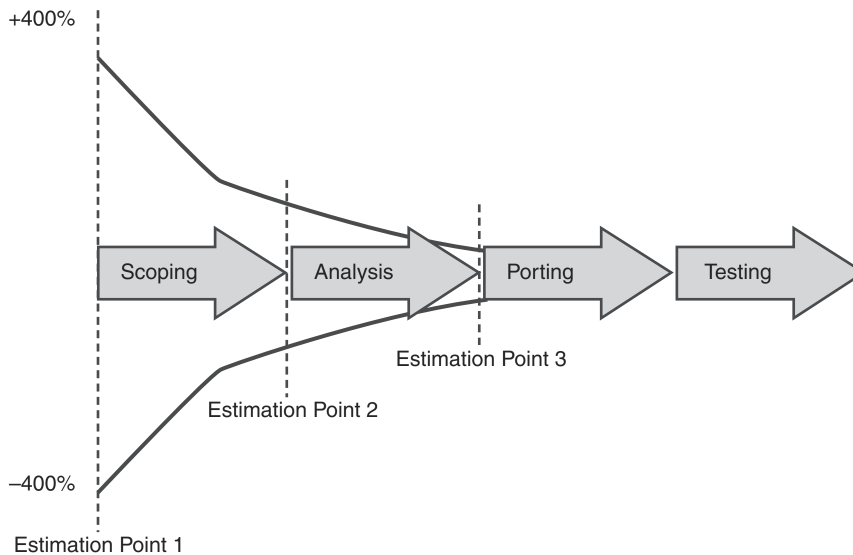


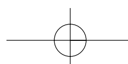

FIGURE 1-2
Average estimation error made before scopes and objectives are cleared

Each transition to the next step in the porting process allows the project team to reassess the project estimates in terms of the schedule and resources. In essence, these transitions can and should serve not only as milestones for the project but also as checkpoints to reexamine the original estimates.

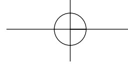
## 1.6 Porting Process from a Business Perspective

The porting process is not just about porting the software application; the resulting ported application needs the necessary business and support structures to make it a successful business initiative. Although porting efforts are taking place, the stakeholders need to prepare other organizations that will provide support for the application. Organizations such as customer support and software distribution will require documentation and training on the application running on the Linux environment.

Getting the necessary Linux training for personnel in customer support and software distribution should be high on the priority list of objectives. As with any new product, new releases tend to generate the most questions from users and system administrators alike. Expect to answer system administration questions about Linux, too. Our experience shows that newly ported applications on a new



Mendoza_01.qxp 3/22/2006 8:51 AM Page 18





operating environment generate questions about system administration and the use of the new operating system in three out of five support line calls.

As system administration and installation questions begin to subside, more technical questions about the application will arise. Support organizations need access to live test machines to duplicate customer problems. These machines may differ from development or porting machines depending on the application and technical problems that will require instances of the application to be running for debugging purposes.

From an overall project perspective, the need to train support personnel and the availability of physical resources to support the ported application will come at later stages of the porting project. As the technical aspects of the porting project near completion, the business aspects of the project pick up.

## 1.7 Annotated Sample Technical Questionnaire

This questionnaire, based on which other scoping questions can be formed, serves as a guide to the porting technical leader. The customer in this case is an internal or external organization that needs to port its application(s) to the Linux platform.

### 1.7.1 Platform-Specific Information

1. What is your current development platform for the application?

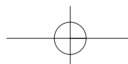
   This question is about the development platform used to develop the application. It does not assume that the development platform is the same platform the application is deployed on. This is asked in the next question.

2. What platform does the application currently run on?

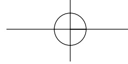
   Porting engineers need to know which source or reference platform the application to be ported is using.

3. Has this application been deployed on any other platform than the development one? If so, which version of the platform is it running on?

   Asking this question gives you a sense of the portability of the application if it has been ported to other platforms. A word of caution: Although an application may have been ported to other



Mendoza_01.qxp 3/22/2006 8:51 AM Page 19





platforms, it may have been done on older versions of the platform.

4. Describe any hardware (that is, graphics adapters or cards) used by your application and whether the required drivers are available on the Linux platform.

   Make sure any platform dependencies are available on Linux.

### 1.7.2 Application-Specific Information

1. Please describe your application and its architecture in detail.

   This is where the customer describes the application architecture. Have them include an architectural diagram if possible. All components in the application need to be described. This should also tell you what type of application framework, if any, the application runs on. Most Java applications run on product-specific frameworks such as WebSphere or Weblogic. If they are written in C++, they may run on a CORBA framework, which means you may have to deal with a specific CORBA framework that may or may not exist on Linux.

2. What are the different components of your software? Please provide a name and version number for each component.

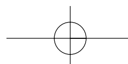
   This gives you a sense of the breakdown of their applications. Being able to break the application into different discrete components can mean that the porting work can be broken into smaller independent tasks.

3. Which of these pieces needs to be ported or not ported? Please include the version number.

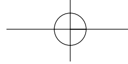
   The customer needs to tell you what is in scope and what is not in scope.

4. What percentage of the application(s) to be ported is/are written in the following programming languages?

   a. Java
   b. C#



Mendoza_01.qxp 3/22/2006 8:51 AM Page 20





c. C
d. C++
e. Assembler
f. Visual Basic (Microsoft)
g. Shells (ksh, csh, perl, awk, others)

Ascertains the complexity of the application by asking what languages they are using and what percentage of those are in use.

5. Please provide a rough estimate of the number of lines of code in the software, listed by language types.

   This is another way of asking Question 4. Asking questions in different ways often brings up data points that contradict each other. This opens the door for open discussions that can result in better project scoping.

6. For Java apps: Does the application use the JNI[9] to link native libraries? Please describe.

   Ascertains the complexity of the application to be ported. Most of the time, Java applications[9] that are not 100 percent pure Java need platform-dependent routines that can only be handled in native language such as C. Be aware of such platform-dependent code, because it takes more time to port.

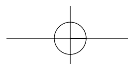
7. Does the application use kernel modules? If so, please describe.

   Ascertains the complexity of the application to be ported. Kernel modules and routines used by the application are nonportable. These will take more time to convert to equivalent routines in Linux.

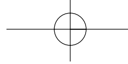
8. Is this a 2D/3D graphics application? Please describe.

   Ascertains the complexity of the application to be ported. Make sure compatible graphics toolkits and development tools are available in Linux, whether they are supplied in Linux by default or by other third-party vendors.

9 Java native interface.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 21





9. Does the application use UNIX pipes, message queues, shared memory, signals, or semaphores? Please describe.

   Most of these have standard UNIX interfaces that can be easily ported to Linux. Although the interfaces may be standard, implementation underneath the covers will differ. The catch is to make sure that the intended behavior remains the same when ported to Linux.

10. Is the application, or any of its components, multithreaded? If so, which threads library is currently being used? Does the application rely on any proprietary threading priorities on your development platform?

    Depending on the source platform, multithreading interfaces can vary from standard to nonstandard. Linux supports several threading libraries, but the one that is standard in present and future Linux distributions is the Native Posix Threads Library (NPTL) implementation. NPTL is discussed in other sections of this book. The point is that the closer the source implementation is to NPTL, the easier it is to port.

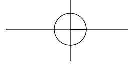
11. Does the application perform operations that assume specific byte storage order? Is this likely to be an issue during the port?

    This question relates to the "endian-ess" of the application. Most Linux ports will target the Intel platform, which is small-endian, whereas most source platforms will be big-endian. Nonportable code that assumes endian-specific characteristics will break when not ported correctly. Worse yet, the ported code will not exhibit errors during the porting phase. The problem usually crops up during system testing, where it is harder to find.

12. Which compiler(s) and version are used in the development platform?

    a. GNU
    b. Java (what version?)
    c. Platform (HP, AIX, Sun, Microsoft, NCR, AS/400, S390, True64) compiler? Which platform?
    d. Others (please specify)

Mendoza_01.qxp 3/22/2006 8:51 AM Page 22





Ascertains the complexity of the application to be ported. If the source application uses the GNU `gcc` or `g++` compiler, it becomes easier to port to Linux because the native compiler for Linux is GNU `gcc` or `g++`. Applications that are developed on other platforms and are compiled in their native compilers tend to use native compiler semantics, which must be converted to GNU compiler semantics. C++ applications become harder to port than C applications when the application starts to use C++ features such as templates. Because some C++ standards are implemented differently by different compiler vendors, porting this type of code takes more time than simpler C or C++ code.

13. In addition to the development environment, are there any dependencies on debugging tools such as memory leak debuggers, performance analysis tools, exception handling, and so forth?

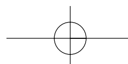
    This goes back to scoping and dependencies. Third-party tools that may or may not exist in Linux need to be assessed. Who needs to provide for the license? Who is responsible for obtaining the tools? What will be the support structure if third-party support is needed?

14. Is this a socket-based application? If so, does it use RPC? Please describe.

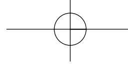
    Although Linux supports standards-based socket and RPC semantics, the intent is to ascertain portability. Asking this question may bring to light nonportable architecture the customer may have implemented in the application. This question can also lead to questions on what setup is needed at the testing phase.

15. Does the application use any third-party software components (database tools, application server, or other middleware)? If so, which ones?

    Every third-party software component adds complexity to the port. If any third-party software components are used, ask what version of the component is used and whether it is available on Linux. Third-party components may require extra time to learn and even to configure or build if necessary.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 23





16. How is the application delivered and installed? Does it use standard packaging? Will the installation scripts need to be ported, too?

    A Linux standard packaging mechanism is RPM. RPM is discussed in other parts of the book. Ascertain whether the customer will need the packaging part of the application ported, too.

17. Is the application or any of its components currently in 64-bit? Will any component need to be migrated to 64-bit?

    With the advent of 64-bit platforms and operating systems, this question pertains to the level at which the application needs to run or be ported. Most 32-bit applications will port to a 64-bit environment without problems through the use of modern compilers. Today's compilers have become efficient at flagging syntax and semantic errors in the application at compile time, allowing the porting engineer to make necessary changes. The one consideration here is that it will require additional time for porting and debugging.

### 1.7.3 Database Information

1. What databases are currently supported? Please include version numbers.

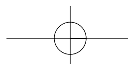
   Almost all enterprise applications today require a database back end. It's important to make sure that the database for the application is available on Linux. Differences in database products and versions can add substantial porting effort to the project.

2. What database is the ported application expected to run with?

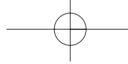
   In addition to Question 1 in this section, what database does the customer expect the ported application to run with on the Linux platform?

3. Does the application use any nonrelational or proprietary databases?

   Any proprietary database needs to be ported to Linux. Make sure the code to run the database is available and is part of the scoped porting work.



Mendoza_01.qxp 3/22/2006 8:51 AM Page 24





4. How does the application communicate with the database?

   a. Programming language(s) (for example, Java, C/C++, other)

   b. Database interface(s) (for example, ODBC, OCI, JDBC)

   Ascertains that programming languages and interfaces are available on Linux, whether or not they are supplied by third-party vendors.

5. Does the application require the use of extended data types (XML, audio, binary, video, and so on)?

   This information can be used to assess the skills needed by the porting group for porting the application.

### 1.7.4 Porting Project Time Schedule Information

1. What is the desired General Availability date for the application on the target platform?

   This question is asking whether any business objectives need to be considered when creating the porting schedule.

2. Has the porting of the application already started on the target platform?

   This is helpful in assessing complexities and problems that are discovered prior to officially starting the porting project.

3. What is the estimated port complexity level (low, medium, or high)?

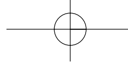
   Take the answer to this question with a grain of salt. There may be other factors present today that may not have been fully realized in previous porting efforts.

4. What factors were considered in this complexity ranking?

   Any information from previous porting efforts needs to be assessed and compared to future porting efforts on the Linux platform.

5. If the application has been ported to another platform, how long did that port take? How many resources were dedicated to it? What problems were encountered?

Mendoza_01.qxp 3/22/2006 8:51 AM Page 25





This question attempts to compare previous porting efforts to the Linux port. This is useful only if the porting engineer technical lead has previous porting experience on other platforms as well as Linux.

6. What is your rough estimate of the project porting time and resources required?

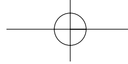
   The application or parts of it may have already been ported to other platforms, and knowing the time it took to port to those other platforms may be of some use. Experience and lessons learned from previous ports may come in handy. Knowing some of the lessons learned may help you avoid problems when porting to Linux.

### 1.7.5 Testing-Specific Information

1. Please describe the acceptance testing setup.
2. What kind of networking and database setup will be required for unit testing?
3. How much testing will be required after porting (number of days, number of resources)?
4. Do you have established test scripts and application performance measurements?
5. Will benchmarks need to be run for comparison testing?
6. Is performance data available on current platforms?
7. When were the performance tests last executed?

All "testing-specific" questions pertain to application software testing on the Linux platform. Asking these questions may bring out other issues related to porting test scripts and the software application test harness, which will add risks and time to the whole project. Pay close attention to the response to Question 1 in this section. Question 1 relates to the type of acceptance criteria that needs to be agreed on before porting starts. An example acceptance criterion is: Module A and B should pass test suites C and D, without failure. When the acceptance criteria are

Mendoza_01.qxp 3/22/2006 8:51 AM Page 26





met, the port is considered complete, and formal QA tests can then be performed on the application.

### 1.7.6 Porting Project Execution Information

1. Please select one or more options, according to how you would like to proceed with this project.

   a. Technical assistance will be provided to the porting engineers as necessary.

   b. Customer will be responsible for acquiring third-party licenses and technical support.

   c. Other (please describe).

   Add other items in this part of the questionnaire that you need the primary customer to consider. Some issues may relate to staff training or testing the application.

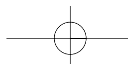
2. What kind of hardware will be needed for the project?

   Consider asking this question to ascertain whether existing hardware may be used or whether extra hardware will be needed—for porting, testing, training, and support purposes if necessary.

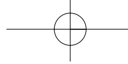
Although this questionnaire is comprehensive, it should not be the only basis for scoping. Scoping should also include actual examination of application source code when pertinent. Software application documentation needs to be examined to learn more about the application from a customer usage point of view.

## 1.8 Summary

Project managers and porting technical leads need to consider multiple aspects of a porting project before embarking on the actual port itself. Careful investigation of these aspects has led us to many successful porting projects in the past. Because every porting project is different, these aspects may take on different shapes and forms. Even so, the underlying concepts and descriptions will remain the same. A porting project can be summarized as follows:



Mendoza_01.qxp 3/22/2006 8:51 AM Page 27





- **Scoping.** Ask questions, inspect the code if available, ask more questions and investigate the risks, and develop the schedule.
- **Analysis.** Ask more questions, investigate the risks, and create the schedule based on technical and business issues.
- **Porting.** Set up the build environment, compile, and unit test.
- **Testing.** Set up the test environment, test, and analyze performance. Go back to the porting process if bugs are found.
- **Training.** Train support and sales personnel on the application and operating environment.

Because complexities and risks associated with different technical aspects of the porting project may affect the overall schedule and budget, the transition point between each step of the porting process allows the project team to reassess its original estimates. Reassessing the original estimates helps set the proper expectations of all stakeholders early on before the real work of porting begins. As in all projects, the success of the project is defined not only by the successful delivery of the final product; it is also defined by how well the project was managed to the satisfaction of all parties involved. In the next chapters, we go through each step of the porting process. We present information about Linux that porting engineers can use as they go through the scoping, analysis, porting, and testing phases of a porting project.