Migrating Applications From Proprietary/Legacy Platforms

Preface

With frequent advances in computing technology, and the increased risk of proprietary/legacy operating platforms being phased out by vendors, the focus on renewal, modification, and updating of computer systems is increasing.

Application migration is now a major issue in established IT departments everywhere. Organizations that have spent years developing applications for particular hardware and operating system environments now see these environments as inappropriate and financially impossible to maintain.

However, these organizations have significant investments in legacy applications critical to their business work flows. Migration of critical applications to another platform (typically UNIX based), offers a way to maintain that investment while gaining additional benefits.

Costs savings are typically realized through reduced capital expenditures and ongoing saving in operating costs provided by the newest hardware. Further cost savings are realized by the standardization in hardware, resulting in highly competitive pricing. Finally, enhanced capabilities are possible because of the extensive selection, and wide availability, of modern software development tools.

ISVs are taking advantage of this situation by consolidating the platforms for which they provide support, and why not? To support an application on one platform instead of many reduces costs, and permits the entire company to focus on that one platform, which ultimately benefits the users of those applications.

The introduction of new technologies such as the Internet, e-commerce, development environments, operating systems, networked data storage, and hardware and communications systems have become the leading edge and focus of the industry.

Indeed, these new technologies have changed the concept of information and its usage in worldwide business. Keeping abreast of new developments and utilizing them to the full are essential requirements for maintaining a competitive edge in today's high-power business environment.

While the need for renewal of business legacy systems is clear, the fact remains that, for the most part, they perform satisfactorily and continue to meet the functional requirements for which they were originally developed. The main reason for this is that the functional requirements, which embody the rules and methods of business transactions for the domain of interest, have not changed appreciably.

Consequently, it is a unique attribute of business legacy systems that they contain a vast amount of valuable business knowledge and procedures that should and will be retained in the new systems that will replace them. What is never taken into consideration is the considerable investment in time, effort and money that has gone into making these systems as functional and robust as they are today – the opportunity to retain some of that investment should not be overlooked.

To that end, there are a number of options to be considered; *Replace*, *Re-host*, *Renovate*, *Rewrite*, *Retire*, *Retain*

This paper is not intended to be a definitive guide to application migration, more an insight into the planning required, and some of the many options available to you – if you have multiple applications and/or multiple systems, then it may be that a combination of some of these options is necessary.

Some of the options you may be aware of, some you won't, some you may have never even considered; but they provide a helpful introduction, and will assist help you in preparing a roadmap for moving forward.

What is Migration?

Migrating, Re-hosting, Re-platforming or Porting? Moving an application from one version of UNIX to another, can generally be referred to as ***Porting*** – fundamentally applications written for one version of UNIX could be deemed to be portable, and can be ported to another version of UNIX with a minimum amount of re-working because of the availability of standard programming interfaces which increase the portability of such applications. In many instances, it is the availability of third-party products that represent the greatest challenge when moving from one UNIX to another.

We generally use the term ***Migration*** to describe the re-hosting of an application from a proprietary/legacy/complex platform to a new one (not limited to, but typically UNIX based) with minimal changes to the application business logic and no degradation to the functionality or stability of the application. By applying minimal change, the schedule, cost and risks associated with major changes to the application are mitigated, and the impact on day-to-day operations is minimized.

The majority of legacy applications are written in a "non-proprietary" high-level language such as FORTRAN, COBOL, Pascal, or C; others combine the use of macro assembler code and real-time languages such as Ada. Intuitively, migrating these applications to another platform is expected to be a "cake walk". Project plans are based on doing a re-compile, linking in some emulation software, and performing some quick tests. In a world that totally complied to standards this would be true, but real world situations make this simplistic approach a recipe for failure. Such applications are not considered portable, and will invariably make use of functionality that adds to the complexity of the application such as the use of operating system extensions, special capabilities or layered products that may not be present on the new platform. Use of such extensions has to be replicated, removed or replaced with comparable functions in the new operating environment. In addition to language differences, data conversion may also be required and adaptation to new system procedures for running an application.

Why Migrate?

The choice to migrate begins with the premise that you recognize "value" in your current application(s) and due to several factors, would prefer them to be made available on a newer hardware and/or operating system environment.

Ascertaining what is influencing the decision to migrate plays an important part in determining your planning and future strategy. Migrating an existing application in part or whole, is a viable alternative to rewriting or purchasing an off-the-shelf product. Clearly, in some cases, rewriting or re-engineering is not feasible because of schedule, cost and risk. Similarly, purchasing a replacement product may not provide all the required functionality (the legacy application may even be so specialized that there is no replacement product available).

Platform Related

- End of life of operating system and/or hardware is determining the move
- You want/need to move from a legacy/proprietary platform to an industry-standard platform
- You wish to reduce your IT cost of ownership by consolidating the number of platforms in your organization
- Technical problems with your current platform (software and/or hardware) represent a risk to your business, so a platform change is preferred

Application Related

- You like the applications very much and wish to retain them regardless of operating platform
- You wish to retain the investment made in your applications
- Customizing off-the-shelf packaged software to meet your needs can prove risky and costly
- Your existing applications still provide unique functionality not found in off-the-shelf packaged applications (unless heavily customized, in which case see the previous point)
- The cost of writing a new application is often prohibitively expensive
- Availability of new business applications is reducing as vendors focus on newer platforms
- Your users are demanding an attractive GUI (Graphical User Interface) instead of the existing character terminal-based UI (even though a GUI might be less efficient in day-to-day use)

Development Related

- Skilled resources in your development and/or operating environment are becoming scarce, expensive, or both
- Availability of new development tools is reducing as vendors focus on newer platforms
- You wish to take advantage of new technologies such as relational databases, client/server and the web

What Approach To Take?

There really are only two approaches to application migration, both have benefits and risks, and so great consideration should be made as to which approach best meets the needs of your organization.

Big Bang

- Attempt to migrate everything at once
- Many simultaneous changes being made to several applications can seriously complicate the development, testing, and debugging processes
- Greatly increases risk to the potential success of the project
- Requires a great deal of "down-time" to roll-out, which makes a roll-back all the more difficult
- Typically increases the time and resources required (and hence costs)
- Existing staff need to be immediately productive in the new environment, or supplemented with staff with the required skills
- Many people adopted this approach when they had no other choice, and would now recommend avoiding it if at all possible
- Unless there is no other alternative, we would generally recommend that this approach is avoided

Phased

- Migrate one or two applications (or application components) at a time
- Your developers can focus on one set of issues at a time
- Fewer changes are being made simultaneously, which means that it is easier to identify the cause of problems if/when they arise
- Simplifies/reduces application testing effort
- Can be deployed in manageable steps which reduces the impact to your organization
- Shortens overall development schedules by simplifying each step (and hence reduces costs)
- Existing staff can adopt new skills over a period of time as the application components are deployed
- Phased stages can be implemented "live" bringing earlier ROI (Return On Investment), while also phasing the costs over an extended period of time

Where Do You Start?

Many people have never had to undertake a migration from one platform to another before, so there may not be a great deal of knowledge internal to your organization as to what the actual requirements are. It is therefore important that you understand what is required to help build a roadmap that dictates the future of your applications (and possibly your organization's IT infrastructure).

Management

Migration of any kind raises a number of concerns specific to management. They are all solvable and when addressed they can build team spirit, teach new skills, and leave a more effective organization.

New skill sets are required of organizations moving from one platform to another. While the goal of most activities will be the same as on any operating system, the specific actions one takes is different. Obvious examples involve system and network management, accessing the e-mail system and simple things like using the editor.

One striking effect a migration project has is to highlight the cultural change the IT staff must go through. For example, staff may have significantly different ways of viewing the world that can affect morale and productivity. For example, it is common to hear staff who are used to the VMS world refer to UNIX as a "fancy MS-DOS".

A migration project that changes critical information systems requires coordinating many diverse activities, including such basics as:

- Hardware resources
- Training of the IT staff
- User documentation
- Train the trainers

Other activities such as beta-testing and product release which may not have been done for years must now also be addressed.

One item sometimes overlooked in a migration project is the impact on the existing customers (your users), which can be a significant issue. Both external and internal customers must be considered. Your customers are used to using your software on one platform and now you are moving to another, the first question in their minds will be "are you abandoning us?"

Choice Of Target Platform

Unless you are in a situation where there is a corporate standard or preferred vendor for particular platforms, "Which platform should I select?" is one of the most common questions that is asked when considering the move from one platform to another.

Hardware

The fundamental differences in hardware architecture must be addressed for a successful migration. Again, to use the VAX VMS example, there are obvious differences between VAX and RISC-based architectures, such as the significantly reduced command set and expanded register counts. But, there are other differences equally important but typically overlooked. These include floating point format, parameter passing mechanism, odd byte addressing, and lastly the fact the VAX is a little-endian machine.

Unless the applications are particularly complex in nature, processor/architectural differences are generally hidden from the "average" programmer, but can have serious implications in a migration project. For example, the floating point formats can significantly change results in applications doing extensive calculations, the parameter passing mechanism may prove problematical if using inter-language calls. What is probably the most significant problem is odd byte addressing (in the best case it will be limited to performance degradation while in the worst case can cause total software failure) and "endianism" of the target hardware chosen.

The Endian Problem

Doesn't exist in some platforms while for others it will be the majority of all errors found. Big-endian and little-endian are terms that describe the order in which a sequence of bytes are stored in computer memory. Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first. For example, in a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage (if 4F is stored at storage address 1000, for example, 52 will be at address 1001). In a little-endian system, it would be stored as 524F (52 at address 1000, 4F at 1001).

Endian problems can manifest themselves in a number of ways, most typically in applications written in languages such as C and FORTRAN. Whilst the application will continue to function, results may be somewhat erratic.

Operating System
The major problems of introducing another operating system to an application is found in the usage of low-level interfaces to the operating system. These interfaces include file access, references to operating system intrinsics, and use of layered and third-party products.

Ultimately, deciding what operating system on which to run your application(s) will depend upon the size of system required, which one your applications(s) require to run on, and the price that you are prepared to pay.

Most vendors offer an implementation of the industries most popular on their range of systems:

- **UNIX** – is a robust, commercial-quality operating system that runs on a wide range of RISC or Intel based systems, from workstations to mainframes - IBM AIX®, Compaq Tru64™ UNIX®, Hewlett Packard HP-UX, Santa Cruz OpenServer/UnixWare and Sun Solaris are those most commonly deployed.

- **Windows®** - doesn't really need any introduction; limited to available on Intel processor based systems. The vast majority of Windows systems that are deployed are desktop systems, with some larger distributed systems making their way in to enterprise data-centers. The introduction of the Intel 64-bit processor family will most likely bring with it deployment in larger enterprises. If a vendor offers an Intel based system, then it's pretty much a 100% certainty that it will run Windows.

- **Linux** - an open source, UNIX-like operating system that has been popular within the developer community for some years, and is now finding commercial support by way of major hardware vendors such as Compaq, HP, IBM and Sun. The effect that this has had is for Linux to become more widely accepted as a viable operating system on which to "Bet Your Business". Linux is available on most of the processor architectures offered by vendors today, including all of IBM's platform offerings.

- **Others** - most vendors also offer operating systems other than the more "popular" ones described, and it may be that one of these is a more appropriate platform on which to deploy your new/migrated applications - IBM's OS/400 and OS/390 and Compaq OpenVMS and Non Stop Kernel are prime examples. One of the biggest advantages of these operating systems is that they tend to be much more secure that Windows/ Linux/UNIX.

Development Environment
The key tool enabling software migration is the compiler. Typical migration scenarios depend on compilers providing support for the majority of the application language. But a compiler is like any other piece of software - it has the potential to have errors or inconsistencies. These can occur both on the new platform and, surprisingly, on the current one.

The errors in the host system might have always existed but because of a combination of different errors the application has worked correctly.

The unfortunate flip side of using "standard" high-level languages and reliable emulation software hides some of the dangers of a migration. These dangers are found with changing the hardware and operating system, which forces major differences in the compiler. These three items together bring out previously hidden errors in the application. The migration activity itself can generate errors in any new code introduced.

Planning

The most important thing to remember is that if a migration is planned for, and executed in manageable, quantifiable steps, success can be ensured.

All of the departments within your organization which are likely to be impacted by the application migration, such as management, the application users, and the various IT functions need to be made aware of what is happening, and need to be involved in planning the next steps.

You should inventory your existing environment – documenting what you currently have helps to establish the size, complexity and scope, and will save a great deal of time when dealing with various vendors because inevitably the same questions always arise.

Risk Assessment

Professional judgment must be used when interpreting both the quantitative and risk analysis, and how they contribute to the overall risk of the migration. In general, the risk elements describe a fundamental set of risks that all software migrations exhibit, regardless of their size or application type. The risks in a migration project, however, must be interpreted in light of a project or organization's business environment and specific circumstances. This interpretation should be based on an informed knowledge of migration projects in general, the organization, and the specific migration project being evaluated. The tables that follow provide a means for structuring this interpretation, but are not an end in themselves. If an organization's risk control of a particular risk item is adequate, but differs from the norm, the rationale for the interpretation should be documented. A documented rationale will help clarify the "why" of certain practices.

Applying professional judgment leads to the issue of the "goodness" of the potential migration project. The risk management process does not place "goodness" requirements on the migration process, although it does establish minimal criteria for a "reasonable" process for many projects. The objective of the analysis is to establish processes that are used and can act as a foundation for systematic improvement based on the organization's business needs.

Risk management on a migration project involves managing the five individual components of risk. The answers to these questions are complex and involve the analysis of different risk drivers. A migration plan that successfully addresses these risk components up front will be a migration project that can be delivered within budget, and on schedule.

***Cost* – Will it be within budget?**

A complete assessment is the first stage of any serious migration project. By understanding and detailing the project size and scope, the risk of the project going over budget is significantly reduced. The key issues of the assessment stage is a detailed understanding of:

- The existing corporate and computer environment
- How software is developed
- The technical environment the application exists in
- The target environment

The assessment process provides opportunities for the company to perform a fairly detailed situational analysis and should result in a clarification of its IT Goals and direction. No matter what format your assessment takes, at its completion every work item for the migration should be accounted for and there must be no unknowns left that will affect the final project schedule or price.

***Schedule* ~ Will it be done on time?**
Planning is the second stage in the project. While the activities are similar to a development project, the slant is different. The first major step in the planning process is to provide a technical architecture for all of the software identified in the assessment stage - this is not necessarily a development effort requiring months of planning. Any items without demonstrated solutions must have small prototypes developed and verified.

The second step during the planning stage is for the inclusion of specific risk management activities to control the risk drivers identified during the assessment stage.

- *Risk Avoidance* is used at the beginning of a project to remove the cause of the risk. This may involve descoping the project or changing some significant factor involved.

- *Risk Control* is performed during the migration project and involves intensive management intervention. It may include activities such as parallel efforts or frequent management reviews.

- *Risk Assumption* would involve going ahead as planned, accepting that there is the potential of not meeting the schedule or going over budget

- *Risk Transfer* involves removing the source of the risk from the migration project. This could mean hiring outside consultants, outsourcing parts of the project, or delaying the high-risk elements of the project.

The results of the Planning stage is a schedule with achievable, tangible, and measurable milestones. Three major activities that must be detailed and included in the schedule include configuration management, automated build procedures, and testing.

A rapidly moving migration project depends on the ability to have incremental builds on a daily basis with automated nightly testing.

***Technical* ~ Can it be done?**
The objective of your migration project is to have working applications on a new platform. This is where point solutions conceptualized during the planning stage are implemented. As the build becomes available, tools are used and are integrated, specific re-engineering tasks are performed, and unit tests are performed.

Configuration management and the build mechanisms at this stage must be carefully watched as they can and will cause the project to fall behind schedule and be over budget.

Unit testing of some of your major subsystems should be performed. The unit tests do not need to be complete, but they must ensure that basic functionality is available.

Operational **~ Will it work?**
The *Validation* stage has historically brought the most long term value to our clients. It can and should begin when the corporation has committed to the migration project. The final deliverable of this stage is your application on a new platform with your users trying it, working with it, and using it. The goal is *A System With Equivalent Quality!*

The major advantage a migration project has is also its potentially largest problem! The fact that "The system is already working" provides some very practical opportunities. Specifically a complete test suite can be available before the first line of code is migrated to the new system.

The process we recommend is:

- Develop a significant number of automated and repeatable test cases on the existing working system
- Execute these test cases on the migrated system
- Automatically compare the results removing changes caused by time and dates
- Repeat the comparison process until an acceptable level has been reached

The test methods we develop are based on the functional capability of the application. These cases are developed using the same documentation that is given to the users. The test cases need not be complicated but should be as comprehensive as possible. A rule of thumb is to have every user function executed and every type of data record read, written, updated, and deleted.

We cannot emphasize enough the effectiveness of nightly automated testing. The results can be analyzed each morning and changes can be introduced and the tests rerun.

Support **~ Can it be maintained?**
The final stage—*Productization*—is the most difficult to complete, as the tendency is to continue to make changes. However, its purpose is clear. The productization stage converts, for example, a "HP 3000 MPE application running on UNIX" into a true "solution". This statement of purpose represents activities that range from dealing with performance issues to adding to a complete client/server GUI.

We recommend during the Productization stage that a performance measurement plan be developed. The performance plan is based on selecting and measuring a typical transaction. The transaction may be composed of a number of individual transactions.

The performance plan provides two significant benefits. The first is that it enables performance improvements to be rapidly introduced and the effect measured. The second benefit is in hardware selection and capacity planning. Using the results of the performance measurement plan selecting new hardware configurations becomes easier and less error prone.

Testing

One of the most important aspects of any migration is testing. If you have no means of verifying that the migrated application works in the same way as it did on the original platform, then how do you know if the migration has been successful? One important thing to remember is that if you choose to change the way in which your application works when it is migrated to a new platform, the test plans will need to be altered accordingly – this is often overlooked during the planning phase.

If you don't have any formalized testing procedures, reviving those old Y2K test plans is a good start. One important consideration is that any test plans that you do have will not be designed to exercise subtle problems that arise out of a platform change such as byte ordering, or the use of new languages and compilers which have their own little "nuances". Testing is even more important when it comes to ensuring that data that has been migrated from the original platform has not been compromised. Application tests should not be used to verify the integrity of migrated data; specific unit tests should be employed for each data file.

The definition and creation of a viable test plan is critical to the success of a migration project. Used during various stages of the project, test plans serve to validate the baseline system, assist the developers in performing unit and system tests, and provide the framework for final acceptance testing. These test plans also serve as an "investment" because they can continue to be used and enhanced throughout the life of the migrated application.

Specific test plan details vary from project to project, however we employ the same overall approach with the same basic requirements for all project test plans. Generally speaking, the more time spent up front putting together a detailed test plan, the greater the chances for success in meeting project deliverables, schedules and expectations. Like many other similar exercises however, the law of diminishing returns comes into play at some point in time.

The danger lies in developing a plan with procedures so complex or comprehensive that it becomes impractical or too time-consuming to execute efficiently. The challenge then, is to define a test plan sufficiently detailed to ensure application integrity and validity; yet simple enough to allow efficient and repeatable execution.

It is, of course, important to consider that the time required to run the tests is dependent upon the speed of the platform on which the testing is being performed.

Automated software testing can be used to ensure that migration projects come in on time and within budget, as they mitigate against the tendency to test too late in the process. Also, automated software testing reduces the manual labor required to do thorough and realistic tests.

Software testing within a migration project has some special characteristics when compared to testing for new development. Since the specifications for the application are complete at the outset, a migration project can take advantage of the opportunity to test early in the process. In addition, since the application is well understood, test cases and scripts can be developed even before the migration begins.

The use of an automated software testing tool is particularly appropriate since it lowers the cost of regression testing after each build, which in turn avoids errors in production.

In addition, automated testing can demonstrate how the system will perform for the required number of users in terms of response time and other measurements. Predicting response time from the user perspective is critical if a migrated application is going to be accepted by the user community. Also, performance testing gives data to the capacity planning effort.

Migration testing has the following characteristics:

- Test cases and scripts can be developed in a known, stable environment
- User expectations for functionality are known, and tests to verify functionality can be created early in the process
- User expectations for system performance, especially as measured by response time, are known, and can be quantified.

Other Considerations

When planning a migration, don't overlook any other software components that may also be running on your systems that are just as essential to the smooth running of your business as the main application(s). Be sure to include these in your system inventory so that you can determine if replacements are required.

Something that is often forgotten is the ability to read backup media from your old system. Imagine that you have shutdown your old system and fired up the replacement in March. Then in September you discover that there was a serious error made in January, so you need to pull a file from the backup tape…uh oh…the backup tape is in a format which can only be read by the new system. Be sure to plan for this contingency.

Be sure to include sufficient training for your staff in the period leading up to the start of the migration project, during and after. One important thing to keep in mind is the moral of your IT staff — many of them will likely have worked on your applications and systems for many years, making them "the experts" — suddenly being thrust into a whole new world where their years of knowledge count for little, can be both exciting and threatening for them.

What To Do With Your Applications?

Third-Party Applications

If all or some of the applications that you are running are supplied and supported by a third-party (ISV), then you need to contact them and discuss their roadmap for the applications that you are using.

For older systems, it is highly likely that the original vendors are no longer in existence, or that the applications have been "absorbed" into the business of another vendor, who provides only on-going support for that particular application/version. In such a situation, it is possible that they offer an upgrade path to new applications that you could take, or suggest an alternative application that meets your needs.

In some instances, you may find that they offer an upgrade path to later versions of your application that run on other platforms, such as Windows, Linux or UNIX. If this is acceptable to you, then it is a win-win situation, and it means that the route to a new platform is less painful. The only obstacle might be the availability of data-migration utilities that will take your existing data files and re-create them in a format acceptable to the new application.

The situation may also arise where a user is very happy with the particular application that is being used, and so does not wish to upgrade to a newer/revised version on a new platform. In these situations, it may be possible to come to some agreement with the original supplier and have them migrate the older version of the application to a newer platform for you; or if they are prepared to provide you with rights to use the source code, have someone migrate it for you. Alternatively you could suggest that the ISV has someone migrate the application on your behalf.

Ultimately, it is important that your original supplier is the first point of contact, so that you can ascertain the options open to you from them, and whether or not it is appropriate to your organization. Some of the situations described here could prove to be costly or have a high-level of risk attached to them.

In-house Developed Applications

If you are running applications that you have developed in-house, if no third-party supports your applications, or if you want to take this opportunity to implement a new solution that can better support your business, you have a number of options open to you.

Replace

It is likely that when your in-house applications were developed some years ago, it was because there was no off-the-shelf applications available to meet your business needs, which may not be the case now. It may now be an opportune time to take a look in the marketplace and see if there is a ready-made solution available. Replacing your homegrown applications with a ready-made one is by far the quickest way to embrace new technologies, and get the new features and functions that your business demands.

As when performing a rewrite, one of the biggest barriers when replacing a homegrown application with one off-the-shelf, is understanding what the current one actually does. If the application has not been meticulously documented, then the first task is to document the business functionality of the existing application(s). This then becomes the starting point for your gap analysis to determine the replacement. If the new application doesn't do what the current one does as a minimum, then it will not be acceptable to the user community, and represents a risk to your business.

There is an alternative to purchasing a new application, which may not have been considered - Application Service Provider (ASP). An ASP offers you the ability to use an application remotely, without the need to purchase new hardware, new applications, or incurring the cost of IT resources to run it. If cost reduction is a priority, an ASP offering has a lot going for it.

Re-host

Moving your application from one operating platform to another can be as simple as a re-compile, or the most complex task ever envisaged. If your application has been written using a 4GL, then there is a possibility that this 4GL will also be available on multiple platforms, thereby simplifying the effort.

It may be the case that some or all of the development languages that you have used are not available on newer platforms, and so a change of development language will be required. Alternatively you may wish to take this opportunity to move away from a language that is so old that they stopped teaching it 20 years ago, and so finding programmers who can still remember how to use it becomes an increasingly difficult task!

When you are migrating, you need to decide whether you want to re-host on a one-for-one basis, or whether you want to introduce changes to any of the components, such as the user interface or database technology.

Using Tools

It is possible to migrate your applications to a new platform by using software tools that will allow your application to function in the new environment. These tools come in a number of guises - those that preserve the look and feel of the original platform and those that move you to the new platform, and expose you to the look and feel of the new environment.

A well-implemented migration methodology allows the user to maintain a single set of source code for the application while it runs on both the original and new platforms. This is particularly helpful for ISVs who must continue to maintain the application for both environments. However, it can be useful to an end-user who has multiple computers and/or multiple sites. It means that all sites are running the same version of the application during transition to the new platform.

It should not be assumed that these tools will provide you with a "silver bullet" – from our own experience, we have yet to find any tools which will guarantee 100% success "out of the box" – we would advise anyone against the thought that you can shutdown your own system on Friday night, migrate over the weekend, and start using your applications on the new platform come Monday morning – this is just so fraught with risk!!

- Software based hardware emulators which permit programs from one machine architecture to run on another by creating a "virtual" machine on the new platform. A typical example of this is the CHARON-VAX family of products which emulate a complete VAX hardware system on an Intel based machine.
- Binary translators which convert a binary executable (that is, compiled) program for one operating platform into a binary executable program optimized for another. An example of this is DECmigrate, which converts a binary executable program for OpenVMS VAX or ULTRIX RISC on MIPS into a binary executable program optimized for OpenVMS Alpha or Compaq's Tru64 UNIX.
- Language translators which can take one programming language and turn it into another, such as our VX/BASIC which takes Compaq OpenVMS DEC BASIC code and turns it into C (because there is no DEC BASIC compiler available outside of OpenVMS).
- Language parsers that take one language dialect and convert it into another. An example of this is our XFORM/COBOL that takes HP 3000 MPE COBOL and converts it into a dialect suitable for compilation by another COBOL compiler such as that from Micro Focus or Acucorp.
- Environmental subsystems which provide functional equivalences for operating system specific functionality such as VMS system services and MPE intrinsics. Generally these tools take the form of linkable libraries which embody the same API structures as those from the original platform, thereby minimizing changes required to the application source code.
- Data extraction, migration and conversion tools which help preserve your most valuable asset. There are many reasons why the data on your existing platform may not be portable due to floating point format, endianess or proprietary file formats. Having the ability to export your data to a new platform and file system without compromise is very important.
- Command shells that replicate the functionality of the command shell from your original host. For many users, when moving from one operating system to another, having a familiar command shell can make them instantly productive in a totally alien environment. Examples of this is our VX/DCL which replicates the OpenVMS DCL command interface, and XFORM/CI which replicates the MPE command interpreter.
- Tools that replicate batch and print spooling functionality can be very useful if your applications depend very heavily on these functions either via command files or through direct application interaction.

Tools, Native, Hybrid or Phased?

There are at least two major classes of users for migration tools. The first category wants to perform a *complete migration*; the second class wants the ability to *migrate on demand*. *Complete migration* users are interested in converting to the new platform and discontinuing use of the original platform. *Migrate on demand* users want to be able to use one source code pool on either the original or the new platform environment. In the first case, once the migration is completed, the tools are no longer needed. In the second case, they may continue to be used for many years.

Complete migration users come in at least two categories; those who are satisfied with *compatibility* of the original run-time environment on the new platform and those who want to run in *native-mode* in the new environment. The quickest, easiest and least expensive method of porting involves the use of a simulated environment that is compatible with the original operating system, hence the name *compatibility mode*. In most cases, this will provide acceptable performance for the life of the system. Some users may wish to convert the software so that it can be optimized to run in the new environment. This native-mode port allows the application to run free of proprietary features from the original platform.

Tools based porting works best for all of the above categories and sub-categories of users. The most important feature of tools-based porting is that the automated conversion can be performed at any time and it can be repeated at anytime. Even if the user wants to perform a complete migration, there is a period of parallel operation. What if during this parallel operation or even after the system is in production, it is determined that there is a major pervasive defect in the software? The solution is to fix the toolset to remove the defect and then re-run the migration. In other words, it's easy and inexpensive to start over! If, on the other hand, the migration were to be done manually, such a change could be frustrating, time-consuming and very expensive.

If you want a complete migration to native mode, tools-based migration is even more desirable. It has all of the benefits mentioned above and had the additional benefit that after the system has been ported to compatibility mode it can be tested and enhanced in a "real world" environment. Therefore, when the last enhancement piece is finished, the system is ready for use a short time later.

Renovate

If you are happy with the business logic components of your application, but wish to breath some life into the user interface and/or file I/O layer, then renovation is a serious consideration. By extracting the business logic from your application, and building into a new framework of software components, a great deal of time and cost can be saved.

One of the major benefits of application renovation is that in many instances, the desired changes can be made, tested and deployed on the original platform to eke some extra life out of the current environment prior to replacement, or they can be performed as a follow-on step from a re-hosting project.

The main areas where renovation can be applied to an application are:

- Convert non-relational (flat/indexed files) to relational database access
- Replace a terminal/character-based user interface with either a graphical (GUI) and/or web based user interface – this generally implies a change from server-only processing to multi-tier client/server processing

Each area of renovation has its own special requirements, requiring different skills, and carries its own risks.

If changing the user interface, be sure to take into consideration the changes required to your deployment infrastructure. If you are currently using dumb terminals connected via serial lines, a change to PC clients that require broadband connections, and a greater level of user support, can often cost more than the application migration.

Rewrite

You've looked around and can't find anything in the marketplace that even remotely meets your needs; your current application just doesn't cut it anymore, what do you do? Think about writing a new application.

Creating a new application from scratch requires meticulous planning, strong organizational skills and a sound technology direction. Be prepared for a project that is going to be behind schedule and over budget. One of the biggest barriers when rewriting an application is understanding what the current one actually does. If the application has not been meticulously documented, then the first task in the rewrite is to document the business functionality of the existing application(s). If the new application doesn't do what the current one does as a minimum, then it will not be acceptable to the user community, and represents a risk to your business.

Modern rapid application development (RAD) tools that make use of the latest technologies (such as Java, XML etc.) can provide a major boost in getting a new application deployed quickly. Typically, many organizations that are maintaining applications developed many years ago (or by others), are unlikely to have the resources with the appropriate skills in-house to develop a new application with functionality that equals or betters the existing one.

You should not embark on a development project until you fully understand all of the risks and challenges involved – only then can you be sure that it will be successful. The obvious benefits of rewriting are that your applications are no longer constrained by the original operating platform, and the you can take the opportunity to fully integrate new levels of functionality that would otherwise have been impossible without major development efforts.

Retire
Will there still be a need for the application that you are currently running over the next 3-5 years? If not, then why not spend the time you have now looking at what you are going to replace it with, instead of figuring out how to migrate it in the short term. There may be another application available on the new platform right now that can take over the functionality that this application provides, allowing you to retire the current application and deploy the new platform sooner rather than later.

Retain
If you think that your application is solid and the platform that you are running on is not currently causing you any problems, the immediate thought is to just leave everything "as is" – the old adage "if it ain't broke, don't fix it" still applies.

In some instances this may be the most appropriate decision for your organization; BUT, if you are using an operating system and/or hardware platform for which the vendor has set an end-of-life date, prolonging a decision may put your organization at risk.

Summary

As has been discussed, migrating your existing applications to another platform is not a given. Before you can migrate, you have to plan, which provides you some insight into the schedule, costs and risks associated with migration – ultimately it is these factors that will influence your decision.

Your needs will be unique, and so comparing your chosen path with that of others is inappropriate; it may be that for various reasons you need to move off of your original platform and onto another, with no loss of function or performance, as quickly and as affordably as possible; others may have chosen a different path, don't be swayed by them, they are not responsible (nor accountable) for the success or otherwise of your business!

If you are unsure as to which approach is best for you, by at least making the necessary preparations now, you can delay an immediate decision and reevaluate your options at a later date.

Migration is the solution to organizations looking for a low-risk introduction to a new platform, with a quantifiable ROI. It is the type of project which on the surface is very straightforward, but when ones looks "under the hood" is actually quite complex. However, if care is taken, with proper planning, migration projects can be delivered on time, within budget, and will meet user and management expectations. If used appropriately by IT management, migration of key applications can lead to a rejuvenated IT organization making better use of limited financial resources.

Sector7 USA, Inc.
6500 River Place Boulevard
Building II, Suite 201
Austin, Texas 78730
United States of America
Tel +1 (512) 340-0606
+1 (800) VMS-UNIX-NT
Fax +1 (512) 340-9777
Email sales@sector7.com

Sector7 (U.K.) Ltd.
Canberra House
CorbyGate Business Park
Corby, Northants NN17 5JG
United Kingdom
Tel +44 (0)1536 408588
Fax +44 (0)1536 408518
Email euro.sales@sector7.com