Realities Of Upgrading Large Real World Applications

Introduction

Should I upgrade or rewrite? How long will it take to upgrade my application? What are the most common upgrade issues?

In this document, we answer some of the most common questions about upgrading. The following answers come from our experience in upgrading large real world applications. We've successfully upgraded a wide range of applications from Visual Basic 6 to Visual Basic .NET; including banking systems, enterprise resource planning software, graphical reporting tools, CASE tools and healthcare information systems. In all, more than a million and a half lines of code. While every application is different, some trends have emerged with remarkable consistency; and while every upgrade is different, we hope the general trends we have found will help you with upgrading your own projects. So: should I upgrade or rewrite? The answer may surprise you.

Should I Upgrade, Re-Write Or Leave In VB6?

When asked this question, many developers will invariably say "re-write", since they usually feel they can write it better the next time armed with the benefit of hindsight. Certainly if the application is poorly designed, re-writing is can be a good option since it provides an opportunity to 'do it right'. However, examining the business case for upgrading, rewriting or leaving in Visual Basic 6 provides some interesting insights.

If the application already supports your business needs, doesn't need enhancements, and if you already have support staff trained in Visual Basic 6, then leaving the application in Visual Basic 6 is a good option.

If there is a business need to move the application to Visual Basic .NET, then we need to look closer at upgrading vs. re-writing. A popular reason for moving an application to Visual Basic .NET is to either web-enable the application, or enhance an existing web-enabled application with ASP.NET features such as tracing, flexible state management, scaleable data access, and improved performance. As mentioned previously, re-writing sometimes yields an improved application. The downside is that the development cost will be much greater than upgrading. A good Visual Basic

programmer writes between 500 and 1000 lines of deployable code per month. The same programmer upgrading an application will process between 35,000 and 40,000 lines per month. Upgrading is 35 to 80 times faster! The main reason for this huge difference is that a re-write involves writing, testing and debugging a new algorithm, whereas an upgrade means simply ensuring code runs as it used to.

There are benefits to re-writing. Re-writing allows you to correct a previous poor design, and COM objects can be replaced with .NET objects which are more scaleable and don't require registration during deployment. The flip-side of this is that upgrading is much quicker; COM objects can easily be replaced with .NET objects after upgrading; and a well designed Visual Basic 6 application upgrades naturally to a well designed Visual Basic .NET application.

How Long Will It Take To Upgrade My Application?

Because every application and every upgrade is different, and because developers write code at different speeds, there is no generic answer to how long it takes to upgrade an application. However, data collected from our consulting practice forms the basis for the figures below. Please be aware these are merely guidelines, your results will be different.

An 'experienced Visual Basic .NET developer' upgrading a well-designed application can upgrade and test between 10,000 and 15,000 lines of code a week. So if your application is 30,000 lines of code, it might take two weeks to upgrade. We use the term 'experienced Visual Basic .NET developer' because experience plays a big role. Many people upgrading are new to both upgrading projects and to Visual Basic .NET. For a Visual Basic 6 developer learning both skills at the same time, it may take several months to be an 'experienced Visual Basic .NET developer'. Generally, people new to .NET are productive at upgrading after two weeks, and able to upgrade 7,000 lines of code a week after a month. After four months of experience, they can upgrade 10,000-15,000 lines a week. When they start adding new features to the upgraded project, as in any new development their productivity drops to 100-200 lines per week.

These are rough guidelines. Your mileage will vary. Being familiar with the application you are upgrading will speed up these times.

What Kinds Of Issues Will My Project Have?

How many issues will there be?

As in the previous question, every application is different. It is impossible to know how many and what kind of upgrade issues your application will have. However, we can provide some general statistics based on a large number of upgrades. Please note: all statistics in this section are from projects that were first prepared in Visual Basic 6 for the upgrade to Visual Basic .NET. If you don't prepare your application in Visual Basic 6 first, then expect your actual numbers to be much higher.

The five most common upgrade issues together generate 88% of the total quantity of issues. Here they are, with the percentage each generally occurs:

Couldn't resolve default property of object 52%
Property/Method was not upgraded 13%
Property/Method/Event is upgraded but has a different behavior 12%
COM expression not supported 7%
Use of Null/IsNull() detected 4%

These are all micro issues. Micro issues are commonly occurring easy-to-fix problems.

41% of the issues occur in form design, 59% of the issues occur in modules, classes, code-behind-forms and other project items. On average modules, classes and code-behind-forms have one upgrade issue for every 106 lines of code; forms have one upgrade issue for every five controls.

If I Prepare My Application In Visual Basic 6, Will The Upgrade Then Be Quicker?

If it is quicker, how much quicker will it be?

Almost without exception, if you take steps to prepare your application in Visual Basic 6, the upgrade will be quicker. The preparation steps are well documented in other papers, so we won't cover them here. The best way to determine what to prepare in Visual Basic 6 is to run the application through the upgrade wizard, use the report it generates to identify things that can be fixed in Visual Basic 6,

make the appropriate changes, then upgrade the application again. This is commonly called a 'two pass upgrade'.

How much quicker will it be? It depends on the characteristics of the code: for example, a single variable defined as a variant will generate an upgrade issue each time the variable is subsequently referenced. This can amount to a lot of issues. As a general rule, we find that for every upgrade issue you resolve in Visual Basic 6, it prevents 5-8 issues from happening in Visual Basic.NET. Prevention is definitely the best cure. As a result, during a typical upgrade our consultants spend 30% of their time preparing the Visual Basic 6 for the upgrade.

Can I Web-Enable My Application After Upgrading?

Many developers are moving to Visual Basic .NET to web-enable their applications and take advantage of XML Web Services. Web-enabling an application is a great reason to upgrade, since it is both simple and quick.

The three most common techniques of web-enabling are:

1. Expose business objects as XML Web Services
2. Consume XML Web Services
3. Create a Web interface using ASP.NET WebForms

The basic concept for each technique is the same: the application should already be in an n-tier or componentized state. Ideally, the existing Windows user interface is an EXE, and the business logic is in one or more DLLs.

To expose business objects as XML Web Services, you should either create a new DLL, or add a class to the existing DLL containing WebMethods that expose your application's business logic. For consuming XML Web Services, simply add a method to the upgraded DLL that interacts with both the Web Service and the existing business logic. Doing this means your existing code is the foundation for the web application, and all you're adding is a web transport layer. This is typically a simple and quick task.

Creating a Web interface using ASP.NET WebForms is also a straightforward task. Create an ASP.NET WebForm application, add your upgraded business logic DLL as a reference, and use it from the WebForm application.

One consideration for web-enabling an application is how to implement security: are you exposing your business to the web? How can you secure this information? Fortunately, the .NET platform offers a number of security features such as rich authentication classes, encryption, and code access security. These can be easily added to the upgraded application.

I'm A Beginner. How Should I Learn How To Upgrade?

If you are new to upgrading and Visual Basic .NET, we recommend you start by upgrading a small project first. Choose a simple application with five or less project items (forms, classes or modules). Until you're experienced, avoid projects with areas that have no automatic upgrade path: DAO/RDO databinding; ActiveX DHTML pages; add-ins; applications with memory tweaking functions.

Learning to upgrade to Visual Basic .NET is like learning anything else; starting simple means you are learning in manageable chunks. Also, choosing a simple project means you'll have it up and running quicker, this gives a good feeling of progress.

When you're ready to try a larger real-world application, choose one that will benefit from the new features of .NET. For large applications, the principle of divide and conquer applies. The best approach is to upgrade it module-by-module. Start with the client, and then move up the dependency hierarchy, upgrading the business logic and data tier. If possible, test each module of the application as it is upgraded so you know that you are on solid ground before moving forward.

When In The Product Lifecycle Should I Upgrade?

Do I also have to upgrade the business data to Visual Basic .NET?

The best time to upgrade is when the application codebase is stable, but is due for some enhancements. This way, you can combine the upgrade with feature enhancements.

We recommend that you do not attempt an upgrade if your application is undergoing a period of intense change. Upgrading a large real world application may take several weeks or even months.

During this time, if developers are also making changes to the Visual Basic 6 codebase, then you have to either make changes in both codebases, or re-synchronize the codebases at a later time by upgrading the changed Visual Basic 6 application again. You can't re-upgrade a project without losing the changes you already made in Visual Basic .NET. If the nature of the project is such that you have to work on the Visual Basic application while also upgrading it, we recommend you minimize the disruption by choosing modules to upgrade that are stable.

If your business data is stored in a database, then it will be unaffected by the upgrade – upgrading the application does not affect the database. However, if your application contains user controls, the user controls may serialize custom data to a property bag. In this case, you will have to migrate this data separately to Visual Basic .NET, since .NET uses a different serialization format than Visual Basic 6. A good way to do this is to write a custom function in Visual Basic 6 to write the data to a file and then a corresponding function in Visual Basic .NET to read the data from the file.

Will There Be Any Difference In Performance Between The VB6 And The .NET Version?

After upgrading the performance is usually very similar to that of the original application. You may find .NET applications have different system requirements than their Visual Basic 6 counterparts, often you can improve performance by simply adding more memory to the machine.

Where .NET really shines is in the ability to fine-tune application behavior and performance. You can achieve significant performance improvements by replacing ADO, RDO and DAO data access with ADO.NET since it is a disconnected architecture and designed for scalability. With ADO.NET, more data functions can be done in-memory which in ADO required a round trip to the database. The ADO.NET data reader provides a high performance forward-only connected data access mechanism. For intensive string functions, the System.Text.StringBuilder class is much quicker than Visual Basic 6 string manipulation. Visual Basic .NET also supports multi-threading, so you can assign long-running tasks to a background thread. You can use the counters and tracing classes in the System.Diagnostics namespace to track performance and identify bottlenecks. Finally, moving to ASP.NET provides a huge performance improvement over ASP, primarily because the pages are compiled instead of interpreted.

How Do I Upgrade From Visual Basic Versions 1 Through 5?

The upgrade wizard is designed to upgrade Visual Basic 6 applications. The wizard can also upgrade many Visual Basic 5 applications (since the file formats are very similar), however many Visual Basic 5 ActiveX controls are unusable in Visual Basic .NET Windows Forms. The reason is because these controls use a different threading model than Visual Basic 6 controls; this threading model is not supported in Windows Forms. A simple way to establish whether a particular control is unusable is to try adding the control to a Visual Basic .NET Windows Form. If the control works without problems, then there is a good chance the Visual Basic 5 project can be upgraded. If the control is unusable, then upgrade the project to Visual Basic 6 first.

For projects written in Visual Basic versions 1 through 4, upgrade them to Visual Basic 6 before upgrading to Visual Basic .NET. To upgrade a project to Visual Basic 6, simply open the project in Visual Basic, and save it. If Visual Basic 6 prompts you to upgrade controls to their Visual Basic 6 versions, choose ‘Yes’. As mentioned above, if the project contains Visual Basic 5 ActiveX controls, it is often best to replace these controls with Visual Basic 6 versions. For projects written in Visual Basic versions 1,2,3 and 4 16 Bit, you may need to make extra modifications to the converted project to get it working in Visual Basic 6, since some VBX controls will not be automatically upgraded. You will also have to replace Win16 Windows APIs with their Win32 counterparts.

Visual Basic versions 2 and 3 often require an extra step. Visual Basic 6 can only open files in ‘Text’ format, whereas Visual Basic versions 2 and 3 support two file formats: ‘Binary’ and ‘Text’. Before upgrading these projects, ensure the entire application is saved in ‘Text’ format. To do this, select each file in the project and choose the File|Save As menu item. Check the 'Save As Text' checkbox on the resulting Save dialog. Since Visual Basic 1 only saves in ‘Binary’ format, these projects will need to be opened in Visual Basic 2 or 3, and saved as text before they can be upgraded to Visual Basic 6. After following these steps, you can upgrade the newly created Visual Basic 6 project to Visual Basic .NET.

Why Doesn’t The Upgrade Wizard Upgrade Everything?

When we worked with Microsoft to design the Visual Basic .NET upgrade wizard, the focus was to enable the upgrade of enterprise applications. The wizard’s design is straightforward: where possible, upgrade each Visual Basic 6 element to its .NET equivalent. If a .NET equivalent doesn't exist, upgrade to a compatibility class that offers the same functionality. Where this is not possible,

insert an issue into code linked to a guideline in online help for fixing the problem. For enterprise applications, there is usually a solution that improves over the Visual Basic 6 architecture.

Visual Basic .NET helps developers write better code: more robust, more scaleable, and with more fine-tuning control than in previous versions. Every upgraded application takes advantage of these features. So, although upgrading often requires some modifications on your part, the result is frequently an enhanced application that improves upon the previous version.