Planning a Successful Visual Basic 6.0 to .Net Migration: 8 Proven Tips

Jose A. Aguilar

January 2009

Introduction

Companies currently using Visual Basic 6.0 for application development are faced with the challenge of moving away from a platform that, as of March, 2008, is no longer supported by Microsoft[1]. One option that is available to those companies is to use automated migration technologies to help migrate these application into Microsoft's .NET Framework, the recommended upgrade path. Migrating allows companies to leverage the investment in the current application, while moving into a fully-supported and updated development environment.

There are several tools in the market that can help with this migration process. Microsoft ships the *Visual Basic Upgrade Wizard* with all version of Visual Studio .NET, and companies like ArtinSoft offer the *Visual Basic Upgrade Companion*, a migration solution with a proven record of successful migration projects in the past years.

Even with the help of these migration tools, it is still necessary to plan the migration project in order to ensure its success. ArtinSoft has been involved in automated software migrations for more than 15 years, and has been working alongside Microsoft for more than 8 years performing Visual Basic 6.0 to .NET migrations.

In this whitepaper we present eight recommendations that you should take into account when planning a Visual Basic 6.0 migration to the .NET Framework. These tips are based on ArtinSoft's experience performing Visual Basic 6.0 to .NET migrations.

[1] Product Family Life-Cycle Guidelines for Visual Basic 6.0 http://msdn.microsoft.com/en-us/vbrun/ms788707.aspx

The Migration Project Methodology

Throughout its experience, ArtinSoft has developed a very mature project methodology that allows for a very predictable, controlled migration process. This is achieved by dividing the application upgrade into two main phases: first, getting an application in the new platform that has 100% Functional Equivalence to the original Visual Basic 6.0 application, and second, performing incremental changes to leverage the new functionality available in the .NET Framework.

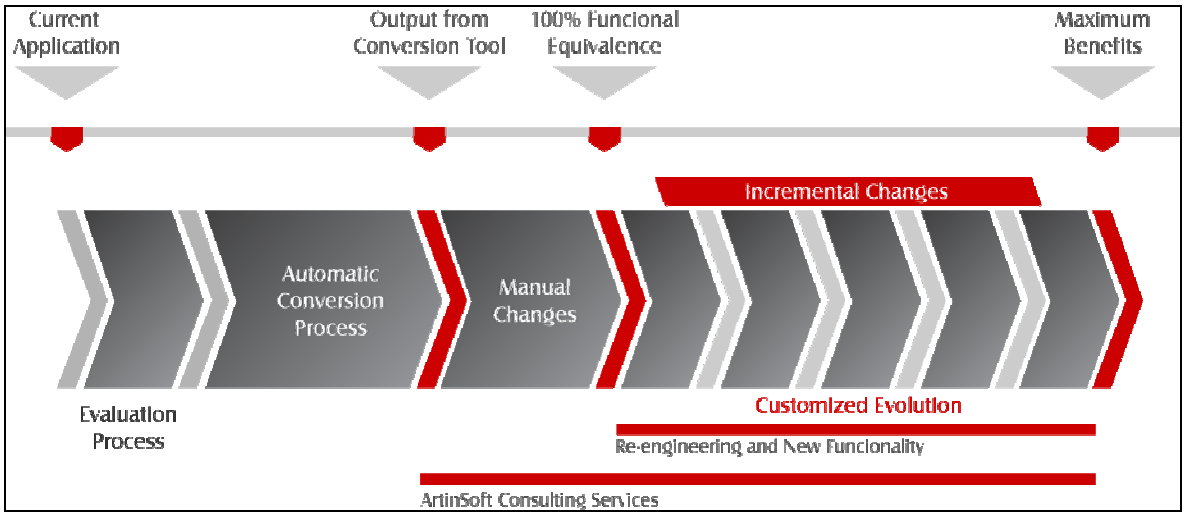


Figure 1 ArtinSoft Migration Project Methodology

With this methodology, the migration project is divided into several process groups:

1. Evaluation Process
2. Automatic Conversion
3. Manual Changes
4. Functional Equivalence
5. Customized Evolution

Each process group is further subdivided in smaller stages. For example, the main objective of the Manual Changes processes is to make any necessary changes upon the code generated during the Automatic Conversion, in order to get it up to Functional Equivalence. This requires several activities, such as fixing the Upgrade Warnings from the automatic conversion tool, fixing compilation errors, addressing runtime errors, and performing different levels of quality assurance activities throughout the process.

The tips presented in this whitepaper are framed in this migration process. For more information, you can access ArtinSoft's website at http://www.artinsoft.com.

Tip #1: Perform Pre-Migration Tasks to Improve the Quality of the Migrated Code

A migration project requires you to plan for some pre-migration work. In this first part of the project it is necessary to perform certain activities to improve the quality of the code that will be generated during the automated migration. There are two types of pre-migration tasks that involve changes to the Visual Basic 6.0 code base: Code Cleanup and Code Preparation.

The Code Cleanup activity is an opportunity to perform maintenance tasks on the code that are usually put off during the regular development cycle. This includes removing code that is no longer used, removing redundant functions or methods, and some basic code and project restructuring.

For the Code Preparation activity the developers should modify the application's code to improve the quality of the generated code. There are tools out there that can help you identify code in the application that will not convert cleanly. One such tool is Microsoft's own Code Advisor for Visual Basic 6.0[2]. Keep in mind, however, that if you are using a tool like ArtinSoft's Visual Basic Upgrade Companion, the migration tool will take care of most issues reported by the Code Advisor[3]. Some issues are easier to fix in VB6 than in .NET, such as the ***Use of #if*** and ***Non-Zero based arrays***, so you should take care of them regardless of the migration solution you are using.

Also, keep in mind the following rule of thumb: *Using high quality code as input for the VBUC generates high quality .NET code as output*.

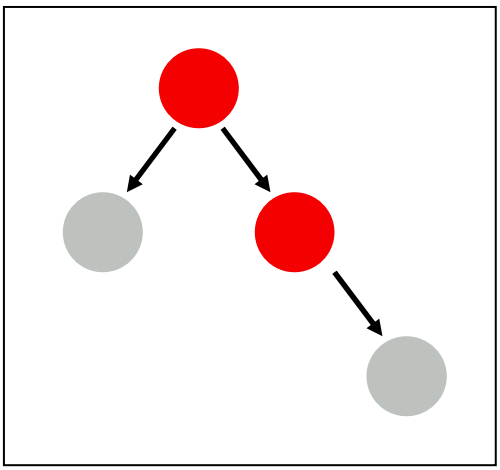
Tip #2: Establish a Migration Order Based on Interdependencies and Business Needs

Migration projects, depending on the company's needs, can be structured in several ways. The two most common ways of establishing this order is to use either a Top-Down or Bottom-Up migration.

---

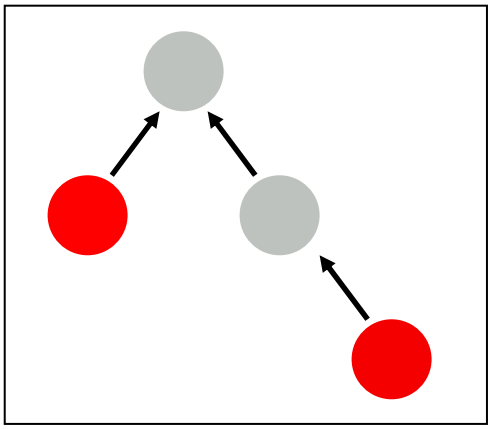
[2] Microsoft's Code Advisor for Visual Basic 6.0 can be downloaded from http://www.microsoft.com/downloads/details.aspx?FamilyID=A656371A-B5C0-4D40-B015-0CAA02634FAE&displaylang=en

[3] Visual Basic Upgrade Companion vs. Code Advisor http://www.artinsoft.com/VB-Upgrade-Companion-vs-CodeAdvisor.aspx



Top-Down migrations, starting with the VB6 projects (*.vbp) you want and then working your way down to the migration issues, are recommended if:

- You need to do partial deployments
- The cost of additional testing and integration work is acceptable
- You are working on a proof of concept



Bottom-Up migrations, on the other hand, imply that you will first fix the upgrade issues in the generated code, and then start moving upwards until you have fully functional projects. The downside of this is that you need to have at least all the migration and compilation issues solved before you can start testing the application.

As for the work distribution between the developers that will be assigned to the project, it can also be done in several ways, each with its advantages and disadvantages:

- Per Visual Basic 6.0 project (*.vbp): This is usually recommended if the original developer of the VB6 project will be working on the migration. Due to the developer's familiarity with the code, it will be easier to fix the different migration issues present in it.

- Per Visual Basic 6.0 file: This is a more scalable approach than going per Visual Basic 6.0 project. Different files can be assigned to different developers and have each one fix all the compilation and upgrade issues present in these files. This can significantly speed up the first stages of the Manual Changes process by parallelizing the work.
- Per Migration Issue: Having a developer specialize in the solution of one particular type of migration issue allows an even higher degree of concurrency in the project. Usually fixing the first occurrence of a particular issue requires some research and thus takes more time. Further occurrences are normally fixed in a fraction of the time it took the first time. By using this approach, the developer's efforts are optimized, and it also lends itself to having a large number of resources tackling different migration issues in parallel, further speeding up the process.

Tip #3: Manage Risks

Even though this is a very generic statement, there are a couple of things that can be done in a migration project in order to mitigate risks. First of all, it is recommended that you convert a small, representative module (using a Top-Down approach) before starting the whole migration. This will be useful to diminish threats associated with:

- **Performance**: The performance of the migrated module can be tested and tuned while performing the migration of the remaining modules. Since VB6 and .NET are significantly different platforms, analyzing the performance upfront will help you identify potential issues and avoid surprises during the last stages of the project.
- **Estimate**: It is necessary to closely track the effort required to migrate this first module. This will help with the validation of the estimate for the remainder of the application

Also, something from ArtinSoft's experience that has been key in performing successful upgrades, is to migrate to functional equivalence first, and then start making changes to leverage the functionality of the new platform. This will be further expanded in *Tip #5: Migrate to Functional Equivalence, then Re-Architect*.

Tip #4: Consider Quality Assurance Activities

Migration projects are more QA-intensive than other types of software development efforts. You should allocate AT LEAST 35% of the total effort of the project to testing activities, though the ideal is to assign around 50% of the time for these tasks.

As for planning the testing, you should use a set of test cases as an objective validation for the migration process. The quality assurance team should make sure that these test cases execute correctly in the Visual Basic 6.0 application before running them on the migrated version. This should be done in parallel by the testing team with the first development activities of the project. This point is covered with more detail in *Tip #6: Identify a Test Harness to Validate Functional Equivalence*.

Tip #5: Migrate to Functional Equivalence, then Re-Architect

Taking the decision to migrate an application means that there is significant value in the business rules and the logic embedded in it. The business may depend on these very specialized rules, so it is imperative to leverage the current investment by moving them forward. This is something very important to keep in mind during the migration project.

It is always tempting, especially for developers, to use the migration as an opportunity to rewrite parts of the applications that could work in a better way. On paper this sounds like the ideal time to do this, but in reality it is something that should be avoided as much as possible. ArtinSoft's experience shows that doing a straightforward port to reach functional equivalence is a measurable, easy way to control the process. Adding additional technical complexity to the project by rewriting a large part of the application adds uncertainty, and can cause the project to go out of control. There will always be an opportunity to do some improvements while performing the migration, but if a decision is made to work on them, make sure that all stakeholders understand the impact that these changes may have on the overall migration effort.

Tip #6: Identify a Test Harness to Validate Functional Equivalence

Migration Projects should have measurable, deterministic criteria to establish when the project is completed. This will set realistic expectations with stakeholders in the project, and in turn will translate into a more manageable and controllable project.

In ArtinSoft's experience, using test cases is an ideal medium to validate when the migration is complete. Having a set of test cases will help the project to have very clear goals: to have the migrated application run the same test cases as the original system, and produce the exact same results.

If there is the possibility of using a third party to provide additional resources during the migration effort, then special care should be taken to make sure that the test cases are as detailed as possible, without skipping any functionality of the application. Keep in mind that these additional resources are

not experts in the application or its domain, so having detailed instructions will allow them to be productive very quickly, without having to undertake application-specific or domain-specific training.

Tip #7: Avoid Introducing Dependencies on Proprietary Runtimes

Several migration solutions out there will release you from one legacy environment only to lock you in with a proprietary runtime. Over time, this approach leads to additional costs, namely:

- Additional support costs from the migration tool's vendor
- Dead time while waiting for the vendor to release a new version of the runtime that fixes an issue that is affecting you
- Possible backward compatibility issues with new releases of the runtime
- Additional training costs and a higher learning curve when new developers start doing maintenance on the migrated software

Using a runtime might be an option when there is a significant difference with the target platform. This may speed up the process, and can significantly lower the cost of the migration, but if you decide to go with a runtime make sure that it:

- Doesn't limit the future scalability of the migrated application
- Provides full source code and documentation, so you don't have to rely on the vendor
- Doesn't have royalties of any kind associated with it

This last bullet is especially important, not from a technical but from a business perspective. Being tied down with redistribution royalties may end up affecting potential business models you may want to explore in the future.

Tip #8: Assemble the right team

The team that is required to achieve a successful migration has a mix of skills and roles. The skills for positions such as quality assurance or project management are very similar to the ones required for any other software development effort. At a high level, the profile of the developers that will be working in the code itself has three main skill sets, in decreasing order of importance:

1. **Target platform experience**. It is ideal that the developers have good knowledge and experience in the .NET Platform (either VB.NET or C#). This is the single most important factor for the migration to be successful

2. **Source platform experience**: It is important to have resources on the team that are knowledgeable on Visual Basic 6.0. This, however, comes in second, and is not required for all developers
3. **Application knowledge**: The ideal scenario is to have the developers of the VB6 application or somebody with in-depth knowledge of the source code involved in the project directly, or at least available for questions. Many questions will arise during the migration, and having an expert on the functionality is the best way to avoid wasting time figuring out what a block of code was trying to achieve.

Conclusions

Using automated migration tools as part of an overall upgrade project methodology is a good way to leverage the current investment in Visual Basic 6.0 applications and move them to the latest technology. Due to the tools that are available in the market today, like ArtinSoft's *Visual Basic Upgrade Companion*, this has become a very viable proposition, especially given the fact that VB 6.0 is no longer supported by Microsoft.

A Migration Project presents some challenges that are not common in other types of software development efforts. With more than eight years executing successful Visual Basic 6.0 to .NET conversion projects, ArtinSoft has developed a proven software migration methodology, and the tips presented in this document are based on the experience accumulated over these years, having proved their value over and over again.