8 Proven Tips for Planning a Successful VB to .NET Migration

Info@mobilize.net
www.mobilize.net

Planning a Successful VB to .NET Migration

Planning is key to a successful migration.

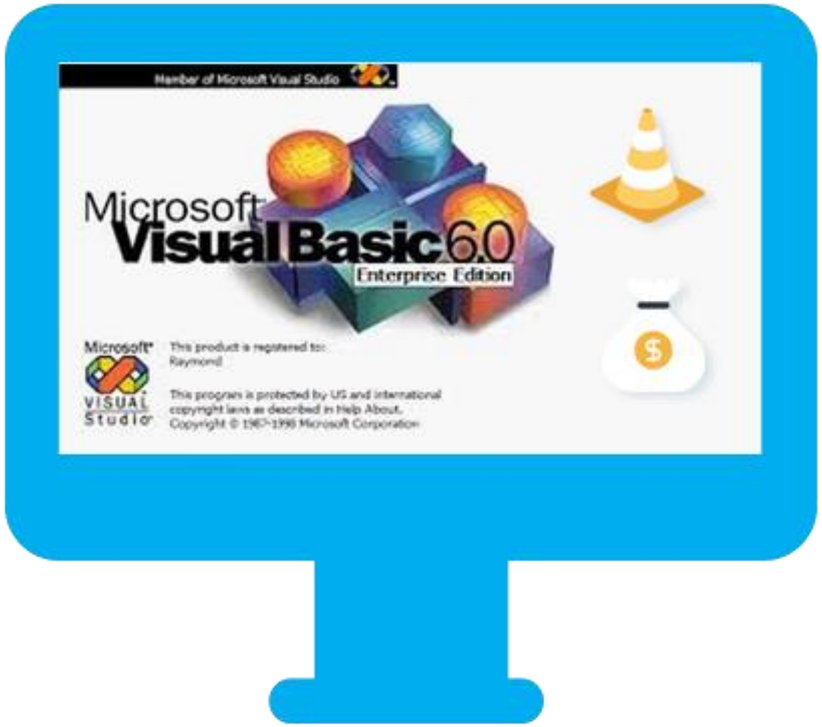
Even with the help of VB to .NET migration tools, it is still necessary to plan carefully in order to ensure success.

This ebook presents eight recommendations that you should take into account when planning a migration to the .NET Framework, based on Mobilize's broad experience executing successful Visual Basic 6.0 to .NET upgrade projects worldwide.

Introduction

VB6 has been out of support for almost 20 years.

Continuing to rely on VB6 is risky and expensive.



Automated migration technologies can help migrate these application into Microsoft's .NET Framework, the recommended upgrade path.

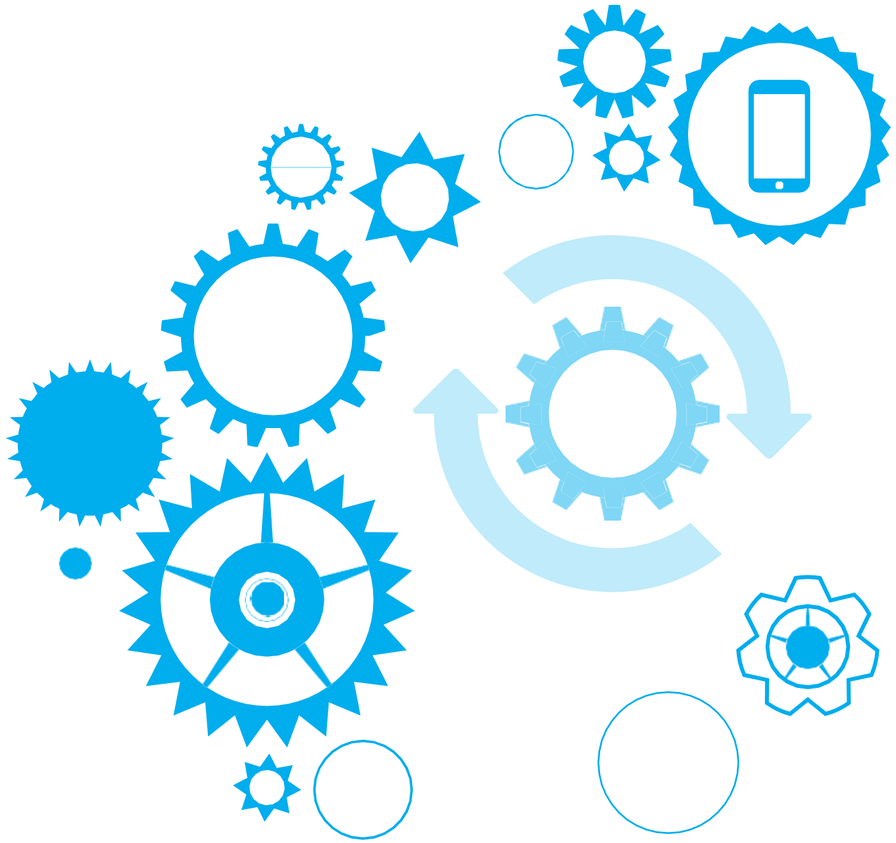
Migrating from VB to .NET allows companies to leverage their investment in the current application, while moving into a fully-supported and updated development environment.

Introduction

Several tools can help with the VB to .NET migration process.

Mobilize.Net offers the **Visual Basic Upgrade Companion**, **WebMAP**, and other migration solutions with a proven record of successful migration projects.

Other companies offer a black box approach with a runtime that requires a customer to pay an ongoing maintenance fee.

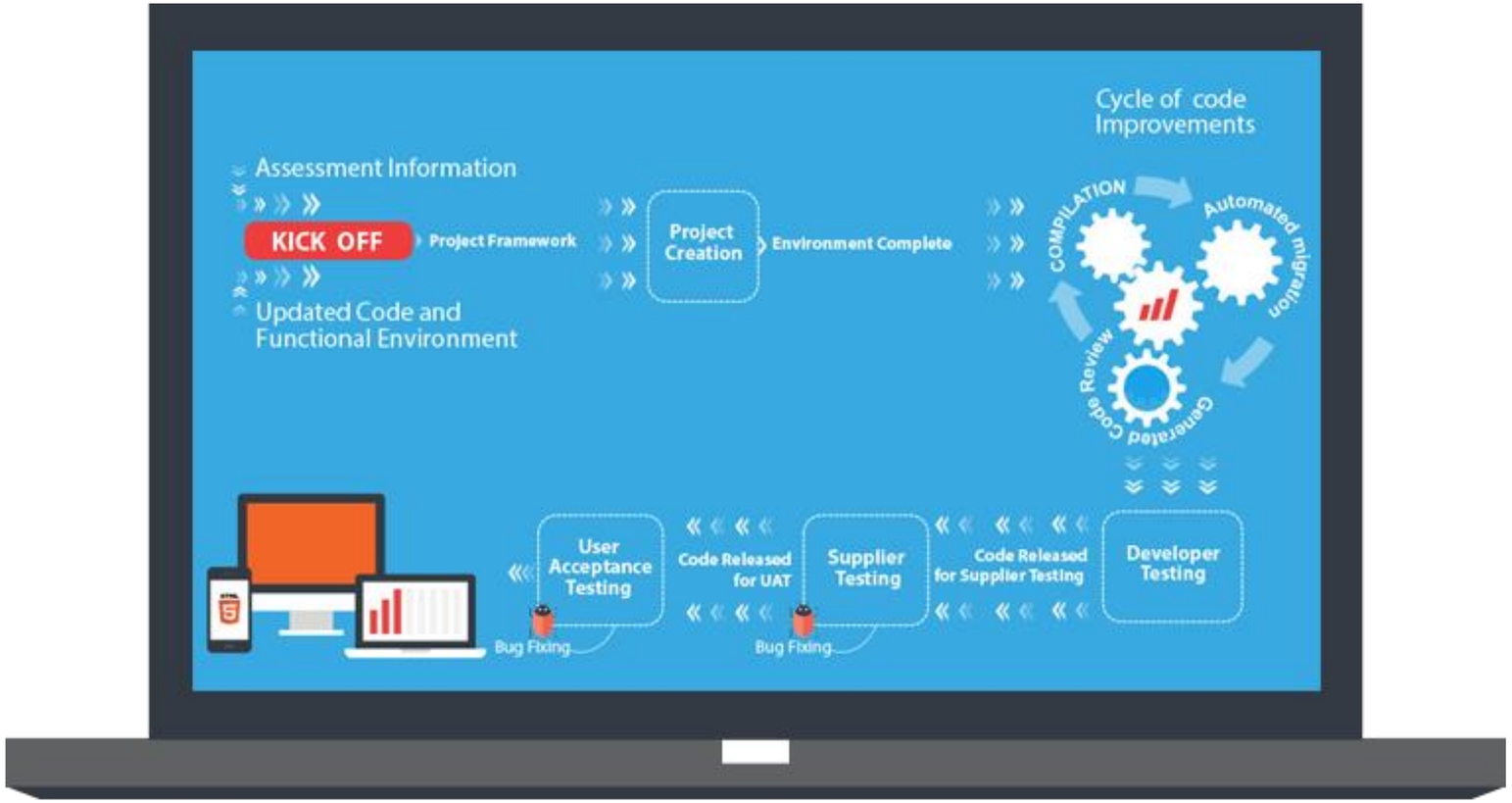


Introduction

Mobilize has been involved in automated software migrations for more than 20 years, and has been working alongside Microsoft for more than 25 years performing VB to .NET & Azure migrations.

No one in the world has more experience than Mobilize.

The Mobilize Project Methodology



Mobilize Migration Project Methodology
With this methodology, the migration project is divided into several processes. This methodology has been successfully used on thousands of projects.

The Mobilize project methodology allows for a predictable, controlled migration process.

This is achieved by dividing the application upgrade into two main phases:

- First, getting an application to the new platform that is 90% functionally equivalent to the original Visual Basic 6.0 application.
- Second, performing incremental changes to leverage the new functionality available in the .NET Framework.

Tip #1: Prepare your code for a better migration

Cleaning up VB6 code prior to the migration reduces the post-migration work.

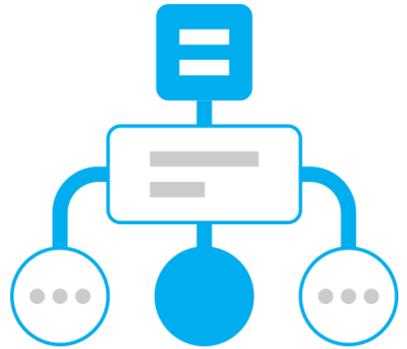
Two types of pre-migration tasks involve changes to the Visual Basic 6.0 code base:
Code cleanup and code preparation.

Code cleanup includes removing code that is no longer used, removing redundant functions or methods, and some basic code and project restructuring.

Code preparation modifies the application's code to improve the quality of the code generated by the VBUC. Mobilize's Assessment Tool for Visual Basic 6.0 identifies many issues that will need attention. Some issues are easier to fix in Visual Basic 6.0 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

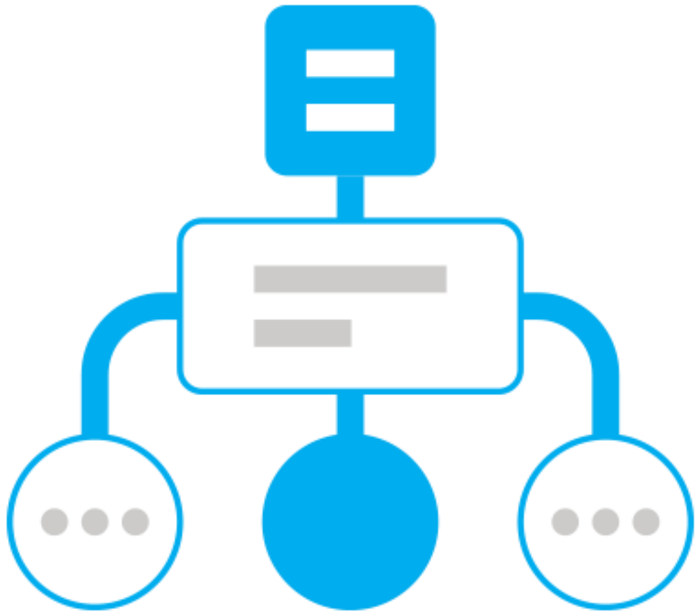
VB to .NET migration projects, depending on the company's needs, can be structured in several ways. The two most common ways of establishing this order is to use either a Top-Down or Bottom-Up approach.



Top-Down VB to .NET migrations, starting with the Visual Basic 6.0 projects (*.vbp) 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.

Tip #2: Establish a Migration Order

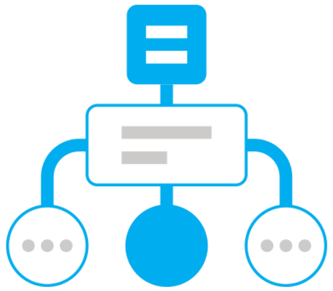


Bottom-Up VB to .NET migrations imply that you will first fix the upgrade issues in the generated code, and then start moving upwards until you have fully functional projects.

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:

Tip #2: Establish a Migration Order

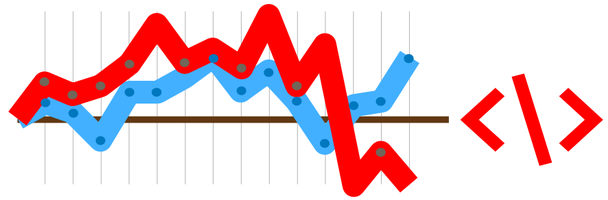


- **Per Visual Basic 6.0 project** (*.vbp): This is usually recommended if the original developer of the VB6 project will be working on the migration. Due to the developer's familiarity with the code, it will be easier to fix the different migration issues present in it.

- **Per Visual Basic 6.0 file**: This is a more scalable approach. Different files can be assigned to different developers and have each one fix all the compilation and upgrade issues found. This can speed up the first stages of the manual changes process by parallelizing the work.

- **Per Migration Issue**: Having a developer specialize in solving 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



While risk management is key to any software project, in a Visual Basic 6.0 to .NET migration project converting a small, representative module (using a Top-Down approach) before starting the whole VB to .NET migration can reduce risk 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 up front will help you identify potential issues and avoid surprises during the last stages of the project.
- **Estimate**: It is essential 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 Mobilize'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

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, 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 Before Re-architecting

Making the decision to migrate an application means that there is significant value in the business rules and logic embedded in it.

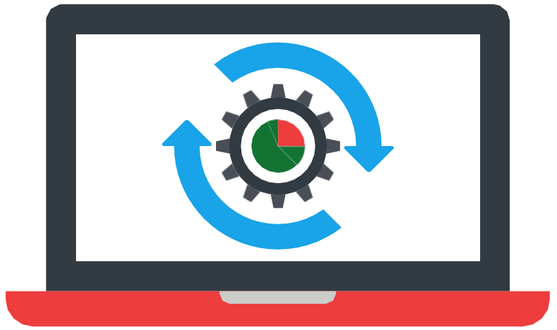
The business may depend on these very specialized rules, so it is imperative to leverage the current investment by moving them forward.

It is always tempting to use the migration as an opportunity to rewrite parts of the application that could work better. On paper this sounds like the ideal time to do this, but in reality it is something that should be avoided. Mobilize'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

VB to .NET 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 Mobilize's experience, using test cases is ideal to validate when the migration is complete.

Having a set of test cases will help clarify project goals: to have the migrated application run the same test cases as the original system, and produce the exact same results.

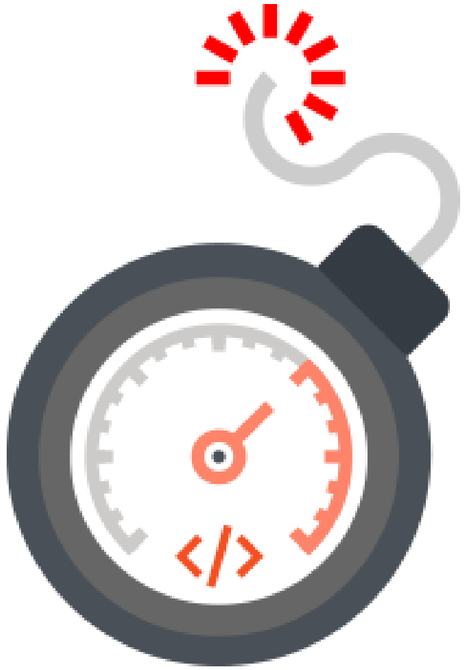
If there is the possibility of using a third party to provide additional resources during the migration effort, then special care should be taken to ensure that the test cases are as detailed as possible, without skipping any functionality of the application. Keep in mind that these additional resources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

Several VB to .NET migration solutions will release you from one legacy environment only to lock you into a proprietary runtime.

Over time, this approach leads to additional costs, namely:

- Additional support costs from the migration tool's vendor
- Lost time waiting for new releases that fix blocking issues
- Possible backward compatibility problems with new releases
- Inability to take full advantage of the new platform
- Additional training costs and a higher learning curve when new developers start doing maintenance on the migrated software.



Tip #7: Avoid Proprietary Runtimes

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 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 affects potential business models you may want to explore in the future.

Tip #8: Assemble the right team

The team required to achieve a successful VB to .NET 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. Ideally the developers are knowledgeable and experienced in the .NET Platform (either VB.NET or C#). This is the single most important factor for the Visual Basic 6.0 to .NET migration to be successful.

Tip #8: Assemble the right team

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: Ideally the original developers of the VB6 application (or someone with in-depth knowledge of the source code) are involved in the project directly, or at least available for questions. Having an expert available is the best way to avoid wasting time figuring out what a block of code is trying to achieve.

Conclusion

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 VB to .NET migration tools that are available in the market today, like Mobilize's Visual Basic Upgrade Companion, this has become a viable proposition, especially given the fact that Visual Basic 6.0 is not supported by Microsoft.

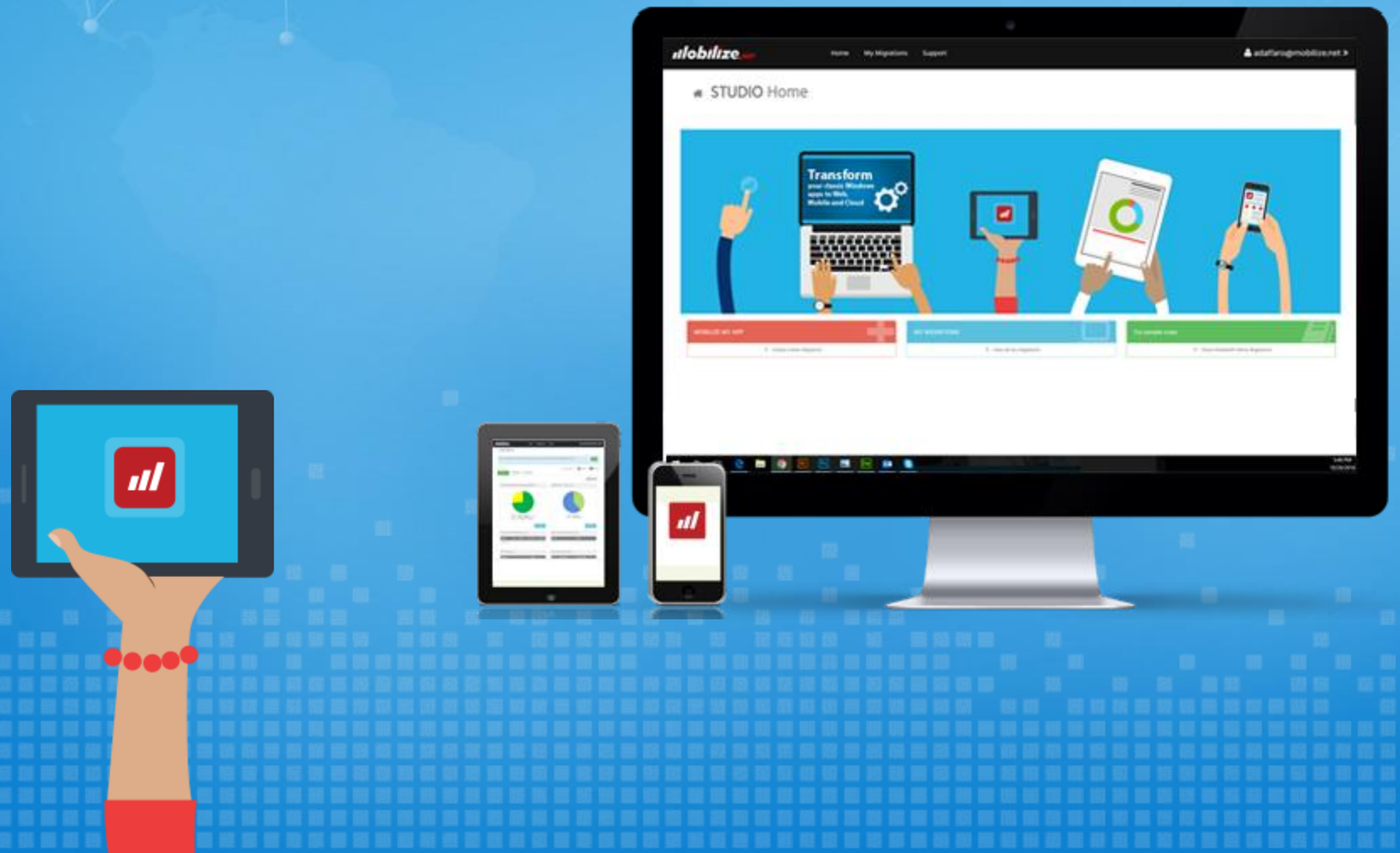
A migration project presents some challenges that are not common in other types of software development efforts. With more than twenty years executing successful Visual Basic 6.0 to .NET conversion projects, Mobilize has developed a proven software migration methodology, and the tips presented in this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

Learn More

Mobilize has assembled resources for successful migrations:

- Kickstart your migration with Mobilize Modernization Starter Kit
- Got Visual Studio? Check out the Visual Studio App Modernization Starter Kit
- Accelerate your move off VB6 with VB6 Migration Jumpstart Guide
- Learn what you need to know with Mobilize Modernization Tech Resources

mobilize.NET



Your App. New. Again.

+1.425.609.8458
www.mobilize.net
info@mobilize.net