Visual Basic 6 to .NET Migrations:
Understanding Performance

April 2023

Visual Basic 6 to .NET Migrations: Understanding Performance

Migrating a Windows desktop application written in Visual Basic 6 (VB6) to C# in .NET carries significant advantages, including enhanced efficiency, improved maintainability, and longevity in the face of evolving technology landscapes. While Mobilize.Net aims to create a more robust and future-proof application with .NET, the performance of the new application depends on many factors: differences in programming languages, architectural modifications, and unforeseen obstacles that may impact performance arising from the hidden aspects of third-party components and Microsoft frameworks and operating systems.

Mobilize.Net makes no warranty that a migrated application will perform identically to the original VB6 application. This whitepaper outlines some of the differences between VB6 and C#/.NET that might affect application performance. While some issues are inherent in the new language and runtime framework, many performance issues can be mitigated with careful profiling and tuning.

1. Inherent Differences in Programming Languages

Migrating from VB6 to C# involves considering various factors that can impact performance. Although Mobilize.Net's migration methodology offers the best like-for-like transformation approach available, the paradigm shift from procedural (VB6) to object-oriented (C#) can lead to variations in algorithm implementation and execution speed. Differences in typing, error handling, string manipulation, default data types, late binding, variable declaration, and event handling between VB6 and C# can also introduce performance variations. Careful consideration and potential code refactoring may be necessary to optimize performance during the migration process.

- **Paradigm Changes:** VB6 and C# are based on distinct paradigms with different design principles. VB6 is more procedural, while C# follows an object-oriented approach. This fundamental difference can cause variations in how algorithms are implemented, thereby impacting the execution speed.

- **Typing:** In VB6, the 'Variant' data type can hold any type of data and is dynamically typed. In contrast, C# is strongly typed, which provides robustness but might cause slowdowns when handling equivalent 'Variant' operations.

- **Error Handling:** In terms of error handling, VB6 uses the 'On Error' statement to handle runtime errors, while C# utilizes structured exception handling, which might increase the overhead in certain scenarios.

- **String Handling:** VB6 strings are mutable and use BSTR data type that provides various built-in operations. In contrast, C# uses System.String, which is immutable. Manipulating large strings can potentially be slower in C# due to the need to create new string instances with each modification.

- **Default Data Types:** VB6 uses different default data types than C#. For example, the default numeric type in VB6 is 'Double', whereas it is 'int' in C#. Any assumptions made in VB6 code based on these defaults could lead to performance issues when translated into C#.

- **Late Binding:** VB6 supports late binding, which allows objects to be dynamically typed and bound at runtime. While C# also supports late binding, it is not typically used due to performance overhead. Code utilizing VB6 late binding would need to be refactored for early binding in C#, which could lead to performance changes.

- **Variable Declaration:** In VB6, variables that are not explicitly declared with a Dim statement are variants of type 'Empty'. On the other hand, C# requires all variables to be explicitly declared before they are used. This can lead to performance differences as variant types in VB6 have more overhead due to their ability to hold any type of data.

- **Event Handling:** VB6 uses a simple event handling model, with events defined within objects. C#, in contrast, uses a more sophisticated event model with delegates and event handlers. This could potentially result in performance differences, particularly in applications with many events.

2. Optimization Complexities

VB6 code, especially those that have been in use for a while, may have been fine-tuned for performance based on the characteristics and runtime behavior unique to VB6. These performance enhancements could rely on specific language constructs or libraries that may not have direct equivalents in C#, making it challenging to achieve similar performance in the new environment. Creating new optimizations in C# to match those in VB6 can be a complex, time-consuming task.

- **Loop Structures and Iteration Mechanisms**: VB6 and C# handle loop structures and iterations differently. VB6 uses traditional 'For' and 'While' loops, whereas C# offers more sophisticated structures like 'foreach', 'LINQ', etc. The loop structure used in VB6 might have been optimized for performance with specific code patterns, which may not translate effectively to C#. Re-optimizing such code in C# to match VB6's performance could be a complex and resource-intensive task.

- **Array and Collection Handling**: C# uses zero-based indexing for arrays, while VB6 supports both zero-based and one-based indexing. Additionally, C# offers more complex data structures like Lists, Dictionaries, and Hashtables. Data manipulation code that has been finely tuned in VB6 might perform differently when translated into C# due to these differences, requiring additional optimization effort.

- **Multithreading and Asynchrony**: The multithreading capabilities in VB6 are significantly limited compared to C#, which offers powerful tools for multithreaded and asynchronous programming. While this is generally a benefit, it could lead to complexities during migration. VB6 code might use certain 'hacks' or workarounds to achieve multithreading, which won't translate well to C#. Additionally, simply rewriting single-threaded VB6 code into multithreaded C# without careful design and optimization could reduce performance due to thread management overhead.

- **Graphics and UI Rendering**: VB6 and C# utilize completely different models for graphics and UI rendering. VB6 uses a simple, immediate-mode rendering system, while C# (through .NET) uses a more complex, retained-mode system. This difference can have a major impact on the performance of any graphics-intensive code. Optimizing graphics code for the new system can be very complex and requires in-depth knowledge of both the old and the new rendering systems.

3. Differences in Architecture

Visual Basic 6 (VB6) and .NET differ in several key aspects. VB6 relies on the Component Object Model (COM) for component handling, while .NET uses a managed and type-safe runtime environment. Memory management, code execution, data access, and UI architecture also vary between the two frameworks, potentially impacting performance during the transition. Careful consideration is necessary when migrating from VB6 to .NET to address these differences and optimize performance.

- **Component Changes:** VB6 uses a different approach to handling components, based on Component Object Model (COM), while .NET employs a managed and type-safe runtime environment. This change in component handling can impact performance.

- **Memory Management:** Differences in memory management and garbage collection methods between VB6 and .NET may lead to temporary performance dips, particularly during the early transition stages.

- **Managed vs Unmanaged Code**: VB6 operates largely on unmanaged code, directly interfacing with the system's hardware resources. In contrast, .NET applications run managed code, where system resources are handled by the .NET runtime, increasing safety and maintainability but possibly affecting the performance of low-level operations that were optimized for unmanaged execution in VB6.

- **Data Access**: VB6 uses technologies like DAO (Data Access Objects) and RDO (Remote Data Objects) in addition to ADO (ActiveX Data Objects) for data access. These technologies directly communicate with databases, providing efficient data operations. In contrast, .NET uses ADO.NET, Entity Framework, and other ORM-based data access technologies, which, while offering enhanced flexibility and maintainability, can introduce additional layers of abstraction, possibly impacting performance during data-intensive operations.

- **UI Architecture**: The UI architecture in VB6 is drastically different from that in .NET. VB6 uses a single-threaded apartment (STA) model where UI components are accessed through a single UI thread. On the other hand, .NET supports a multi-threaded apartment (MTA) model, enabling different threads to access UI components. However, this transition requires careful handling of UI updates and can create performance bottlenecks if not handled efficiently.

- **Component-Based Architecture**: VB6 extensively uses COM components for extending its functionality. The COM model provides direct, efficient inter-process communication, which can be optimized for performance. On the other hand, .NET uses assemblies, which while offering better versioning and deployment, may not communicate as efficiently as COM components, especially if they need to interact with legacy COM components through COM Interop.

4. Impact of Uncontrollable Factors

Migrating from VB6 to .NET involves potential challenges and performance considerations. Unpredictable issues may arise due to reliance on Windows API calls, requiring additional code layers in .NET. Deprecated components and the lack of direct equivalents in .NET may introduce performance overhead, especially when finding replacements or developing workarounds. Compatibility issues with ActiveX controls used in VB6 applications could also impact performance. Additionally, differences in how VB6 and .NET interact with databases may affect performance during the migration process.

- **Windows API Calls:** Unpredictable issues can arise during the migration process. For example, VB6 functions might rely on Windows API calls that may not be directly supported in .NET, requiring additional layers of code that could affect performance.

- **Deprecated Components:** Deprecated VB6 components, or those from third-party vendors, may not have direct .NET equivalents. Finding replacements or developing workarounds could introduce performance overhead.

- **ActiveX Controls:** VB6 applications might use ActiveX controls which may not be supported in .NET, thus demanding complex replacements.

- **Database Access:** The modernized application will need to work with the existing database. However, differences in how VB6 and .NET interact with databases could impact performance.

In Summary

The objective of Mobilize.Net migration projects is to enhance the overall application performance, functionality, and maintainability. However, due to the multifaceted nature of such projects, it is impossible to guarantee identical or superior speed for all processes, despite our efforts to improve performance during the migration.

GAP Mobilize is committed to a rigorous process of testing, optimization, and iterative enhancements to ensure that the modernized application meets the necessary functional and performance requirements. Furthermore, GAP Mobilize has engineering and QA resources trained in C# and .NET to provide support to your team for optimizing and enhancing your migrated application.

This commitment is essential because the migration process involves addressing the complex and sometimes unpredictable nature of the internals of third-party components and Microsoft frameworks and operating systems, as well as the intrinsic differences between VB6 and C#. Our fine-tuned methodology developed over more than two decades helps us ensure that the upgraded application meets the necessary functional and performance expectations.

References:

1. Microsoft Docs, "Accessing Data in Visual Basic Applications". https://learn.microsoft.com/en-us/previous-versions/visualstudio/visual-basic-6/aa716176(v=vs.60)

2. Microsoft Docs, "Choosing Communication Options in .NET". https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

3. Microsoft Docs, "Data Type Summary (Visual Basic)". https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/data-types/data-type-summary

4. Microsoft Docs, "Early vs. Late Binding". https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/early-late-binding/

5. Microsoft Docs, "Event Statement (Visual Basic)". https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/event-statement

6. Microsoft Docs, "Exception Handling (C# Programming Guide)". https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/

7. Microsoft Docs, "Garbage Collection in .NET". https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/

8. Microsoft Docs, "How to: Migrate ActiveX Controls to .NET". https://learn.microsoft.com/en-us/dotnet/desktop/winforms/controls/considerations-when-hosting-an-activex-control-on-a-windows-form?view=netframeworkdesktop-4.8

9. Microsoft Docs, "On Error Statement (Visual Basic)". https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/on-error-statement

10. Microsoft Docs, "Platform Invocation Services". https://docs.microsoft.com/en-us/dotnet/standard/native-interop/

11. Microsoft Docs, "String Class (System)". https://docs.microsoft.com/en-us/dotnet/api/system.string

12. Microsoft Docs, "String Data Type (Visual Basic)". https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/data-types/string-data-type

13. Microsoft Docs, "Variant Data Type (Visual Basic)". https://learn.microsoft.com/en-us/office/vba/language/reference/user-interface-help/variant-data-type

14. S. Clarke, R. J. Walker, "Composition Patterns of Code Migration". https://dl.acm.org/doi/10.1145/2568225.2568285

15. The Computer Language Benchmarks Game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/


GROWTH
ACCELERATION
PARTNERS