SYBASE®

COMPARING SYBASE® POWERBUILDER® AND MICROSOFT® .NET

AN EVALUATION GUIDE FOR POWERBUILDER DEVELOPERS

BY DON HARRINGTON, PRINCIPAL, BLUETAIL LIZARD CONSULTING

WHITE PAPER

TABLE OF CONTENTS

Don Harrington is a developer, consultant, and trainer with more than twenty years of experience in managing the design and development of business and Web applications for technology companies.

INTRODUCTION

Each year development managers are called upon to defend their strategic vision, and the economic repercussions of executing upon this strategy. As visionaries, have they been able to look at the alignment of new technology with the business goals of the enterprise? As decision-makers, can they prove that the actions they've taken resulted in clear, measurable benefits? Development managers must also factor in the input of the developers they lead, and many developers want to broaden their skills and ride the edge of the latest technology wave. As a result, managers are often caught between top-down and bottom-up pressures in choosing new technology directions. No one worth listening to ever said management was easy.

Developers are constantly facing an onslaught of new and evolving technologies. Vendors jockey to differentiate themselves by adding new features that will attract a greater market share. In the case of Microsoft .NET, the huge marketing push behind the debut explained .NET was not simply evolutionary; it represented a new paradigm for the future. Many development organizations grew concerned that they would be left behind if they did not immediately transform into .NET shops. What the immediate converts found in .NET was the latest generation of Microsoft's technology that, while powerful, lacked a spirit of openness and was high on the scale of complexity.

It takes a disciplined perspective to balance strategic vision with economic benefits when choosing to adopt a new developer framework. Canny managers will ask: is it better, does it make economic sense within the context of the organizational needs, and what are the long-term repercussions of embracing the new technology?

This paper examines .NET and PowerBuilder from the perspective of a development manager with PowerBuilder applications. While it is challenging to distill a considered viewpoint from the saturation of .NET information, we will strive to tell the truth. A complete conversion to .NET is not the answer. There is a large PowerBuilder investment in the marketplace and replacing it is not necessarily the most cost-effective action. In the following sections we'll evaluate both the .NET and PowerBuilder technology, the issues for and against a migration from PowerBuilder to .NET and finally, we'll examine the approach of leveraging existing PowerBuilder code and development resources to achieve the best of both worlds with co-existence.

OVERVIEW OF .NET

Microsoft's .NET debuted as a technology preview in June of 2000, hammering the message 'The Internet for Everything.' The definition of .NET was all encompassing, yet so ambiguous that even technically capable consumers couldn't say exactly what .NET was or did. Once the initial impact of the marketing blitz attenuated, the level of lingering buzz was disappointing to Microsoft. Two years after the .NET introduction, Bill Gates was widely quoted as saying "In some respects, we've gotten further ahead than we expected, and in other respects we haven't made as much progress as expected." The relatively slow pace of .NET adoption underscores the point that Microsoft .NET, while a good technology with a huge brand behind it, has not led a fundamental shift in the industry. Successful applications are still being built for the Win32 platforms without taking advantage of .NET, and the unconverted have not withered away.

A general definition of .NET is a program development and execution platform. The programs run wherever .NET resides. Microsoft .NET is a set of technologies targeted at addressing a broad class of applications including standalone, client/server, mobile, smart client, and Web-deployed applications.

Microsoft's development portion of .NET encompasses Visual Studio.NET, the Integrated Development Environment (IDE) that supports application development in various languages. This tool supports several languages, including C# (C Sharp), a language with its roots in C++ and Java™, yet developed specifically for the .NET platform. To continue supporting its legacy development language Visual Basic, Microsoft supplies VB.NET with upgraded Visual Basic libraries. The execution portion of .NET consists of tools such as the Windows.NET Server and services like .NET Passport to support and manage the execution of the deployed applications.

THE .NET FRAMEWORK

The collective assembly of .NET platform components is referred to as the .NET Framework. These components fall into two main areas: the Common Language Runtime (CLR) and the hierarchical class libraries. The Common Language Runtime is the internal execution engine for .NET applications.

One of the unique aspects of the .NET technology is how it handles compiled code. A .NET compiler does not generate code for a specific operating system such as Windows 2000 or XP; instead, it generates code in Microsoft Intermediate Language (MSIL) that is packaged in assemblies. These MSIL assemblies are compatible with the .NET platform. At runtime the .NET platform manages the final step of code generation to the target operating system.

The CLR manages the code conversions from the compiler-generated MSIL code into compatible native code for the actual Windows platform on which the application is executing. Once an executable's code is loaded on a new system, the CLR does not run the final compilation on the whole body of code. It uses a just-in-time strategy of compiling each method within the code into the native system code as it is called for the first time. The CLR is also responsible for loading and executing programs, memory management, cross-language type compatibility, and for enforcing the security properties of the executing code. The other portion of the .NET Framework, the class libraries, provides common functionality to the executing applications as they run.

OVERVIEW OF POWERBUILDER

PowerBuilder is the powerful development environment from Sybase that has a rich history of providing an exceptionally high level of productivity for developers building database-driven applications. PowerBuilder developers create both client/server and distributed n-tier applications. PowerBuilder includes an intuitive IDE, support for multiple external standards, a rich set of libraries, and the unique and patented DataWindow technology. This combination makes PowerBuilder an exceptional OO RAD programming environment.

PowerBuilder is a 4th Generation Language (4GL), Rapid Application Development (RAD) platform. PowerBuilder is an open platform with wide support for commonly used and emerging technology standards, and can easily integrate with components across the enterprise. The 4GL unifies the external technologies and provides a high level of abstraction to simplify complex native interfaces. A programmer with PowerBuilder skills can use any of the incorporated technologies in their code development. PowerBuilder is known for its productivity, which comes from the combination of 4GL abstractions, a mature workflow-driven IDE, and the DataWindow® object. This combination provides a high-level, abstracted environment that hides all but the salient details of developing database-driven applications.

An important feature that differentiates PowerBuilder applications from .NET applications is multi-platform portability. PowerBuilder runs across all Win32 platforms, and when used in conjunction with EAServer, PowerBuilder components can be deployed on a number of Unix platforms.

PowerBuilder's rise to the top of the client-server market was based on its high level of productivity and its ability to work with any database. It is an open, database-agnostic development environment, and provides rich support for both Sybase database technology and other vendors' databases. PowerBuilder applications are frequently deployed with a variety of backend databases including Sybase Adaptive Server®; Oracle®; MS SQL Server, DB2 and Informix, with additional access to a wide range of additional data sources through ODBC, JDBC, and OLEDB.

PowerBuilder plays well with other languages. Developers can easily build applications which make function calls to external DLLs. And, as of PowerBuilder 9.x, a new feature PowerBuilder Native Interface (PBNI), provides the capability to incorporate C and C++ applications into a PowerBuilder application as PowerBuilder objects. PowerBuilder applications can also use the PBNI to interface directly to Java with JNI and call EJBs in third-party application servers. The PBNI also provides the means for Java classes to call PowerBuilder functionality.

Increasingly developers have to support heterogeneous environments. Mergers, acquisitions, and the economic need to preserve legacy investments while adding new application capabilities have led to a mixture of enterprise applications across multiple platforms, each with its own RDBMS. PowerBuilder supports open standards, including J2EE, making it an excellent tool for integrating these diverse data sources.

With PowerBuilder 9.0, applications can build, publish, and consume Web Services. PowerBuilder has a powerful, abstracted XML Document Object Model (DOM) interface. With the XML DOM interface, PowerBuilder developers can parse, manipulate, and create XML to support Web Services and exploit the generalized utility of XML for inter-application data communication, complex data structures, and structured flat files.

Release 10.0 of PowerBuilder enriches PowerBuilder's Web Services support with UDDI search and browse capabilities to enhance the public and private Web Services discovery process. PowerBuilder 10.0 also supports Unicode to simplify the maintenance of user interfaces across different languages, and interactions with databases that store Unicode information. Some additional features of 10.0 are a PowerDesigner plugin for iterative Object Oriented modeling, and the XML-enabled version of the Web DataWindow that supports XHTML with XSL and Cascading Style Sheets to dramatically lower Web traffic requirements.

PowerBuilder applications can be deployed to .NET platforms, J2EE platforms, the Web, and traditional client-server architectures. PowerBuilder functionality can also be re-packaged in Pocket PowerBuilder applications for deployment to mobile users.

PowerBuilder has been the development language of choice for hundreds of thousands of database developers building enterprise applications. Its ease of use and scalability makes it an extremely popular choice for all sizes of projects, ranging from small company applications, to medium department-based applications, to enterprise-wide solutions. PowerBuilder's open development environment is continually being enhanced to support emerging technologies. As developers' needs change, so does PowerBuilder. This ensures that PowerBuilder developers will have an always up-to-date skill set to continue developing the types of applications their users need, based on the newest technologies available.

EVALUATION GUIDELINES FOR POWERBUILDER SHOPS LOOKING AT .NET

Decision makers have become more jaded in evaluating new technology and avoid knee-jerk migrations to the latest product offerings. A careful reckoning of hidden costs and the potential exposure of destabilizing field-tested applications often outweigh the promised benefits of new technology. We'll consider various arguments for and against a migration to .NET.

ARGUMENTS FOR A .NET MIGRATION

In light of the high redevelopment costs, are there compelling reasons to retool an organization with .NET technology? First, let's examine the prevalent arguments for migrating to .NET:

1. .NET REPRESENTS THE LATEST TECHNOLOGY

In .NET, Microsoft created an excellent platform for code development; .NET was well conceived and well executed. Microsoft has also put its best marketing effort behind letting the world know how good .NET is, while seeding a message inferring that development without .NET is somehow inferior and a future disadvantage.

When examining .NET as a potential technology, keep in mind that Microsoft did not develop .NET for purely altruistic purposes and to make programmers multi-functional. They built it to further dominate the marketplace. Latest does not always equate to greatest—there are other viable alternatives that may be more suitable to different needs.

2. MANY PROGRAMMERS WANT TO BE .NET DEVELOPERS

Microsoft .NET is a honey pot for attracting developers with technology that can address many different types of applications. .NET gives a programmer the ability to be a journeyman jack-of-all-trades with the mastery of one.

From the perspective of a development organization, this should not be a compelling argument. The organizational goal is productivity, stability, and – potentially – portability. This could lead to problems down the road as .NET ties a programmer to a single vendor on a single platform with a complex environment.

In reality, many programmers, especially 4GL programmers, will be neither comfortable nor productive in a .NET environment. C# is an excellent general-purpose development language; it also requires an extensive foundation in Object Oriented programming and a senior-level background in architecting enterprise-class applications in a 3GL. .NET's exposure of underlying implementation details increases the complexity of application development. While this detailed exposure adds a fine level of control, it is an impediment to productivity.

3. .NET OFFERS INTRINSIC SUPPORT FOR WEB SERVICES

Web Services is well supported in .NET. It is surprisingly easy to make Web Services extensions to .NET applications. Web Services is an effective method for applications to exchange information locally and remotely using XML as the primary interchange format.

Web Services is not by any means limited to Microsoft. Most vendors, including Sybase, offer Web Services support even though the majority of applications in use today do not yet use them. Objectively, how important is intrinsic Web Services support? Certainly it makes it easier to develop Web Services-enabled applications, but how many Web Services does an application need? Most applications don't need to sprinkle them around like Command Buttons.

When Web Services was initially introduced, many analysts foretold an immediate adoption of the technology in all levels of applications. One of the most compelling ideas behind it was the ability to create public registries of Web Services functionality. Applications could attach to published Web Services by browsing the registry and consuming their functionality. That level of adoption has not occurred due mainly to security concerns; instead Web Services implementations are most often found in trusted application-to-application communication. Public registries are beginning to gain momentum, but far slower than originally envisioned. The upshot is that Web Services-enabled applications are important, but not ubiquitous. Intrinsic Web Services support is impressive, yet it does not automatically put wind in the sails of a development organization. Public application-to-public application security is problematic, and until it is addressed, widespread inter-enterprise adoption is unlikely. Meanwhile, since Web Services are based on XML, existing PowerBuilder applications can utilize them easily

4. .NET CAN ADDRESS A BROADER CLASS OF APPLICATIONS

Microsoft .NET is a flexible technology that can be laser focused on low-level systems programming development and also used in enterprise-wide application solutions. This breadth of development is a compelling argument for .NET.

PowerBuilder was not designed for low-level programming; it was designed for enterprise-scale, rich-client or smart-client applications. The low-level development capabilities afforded by .NET have little if any overlap with PowerBuilder's class of applications because PowerBuilder is an undiluted RAD platform for creating data-driven enterprise class applications. .NET's multi-level adaptability acts as both a strength and a weakness. As an analogy, many people keep a Swiss army knife in their car. Swiss army knives are quite handy as multi-function tools and include a screwdriver, wrench, and corkscrew. However, if you need to sink 100 screws, your first choice of tool would not be the Swiss army knife. Instead, you would want a power tool designed for setting screws. If the goal is rapid development of database-driven applications, .NET does not compete with PowerBuilder because .NET must expose more of the underpinnings of the development environment so it can address a broad class of applications.

5. VISUAL STUDIO .NET IS AN EXCELLENT DEVELOPMENT IDE

Indeed, Visual Studio .NET is a fine development environment. You will find no argument here. For the languages C#, Visual Basic, C, and C++ Visual Studio .NET is an excellent development and debugging environment. Several third-party vendors are also providing .NET versions of COBOL and Fortran. The CLR imposes some restrictions on these languages, but they are supported.

Other than Visual Basic, these are all 3GL languages, which mean greater flexibility and lower productivity. At some point, it comes down to the developer writing lines of code. For most database-driven applications, PowerBuilder—which also has an excellent IDE—supplies the same functionality in fewer lines of code.

6. APPLICATIONS DEVELOPED FOR .NET ARE INTERCHANGEABLE ON ALL WIN32 PLATFORMS

Again, no argument here. For the supported languages, the CLR supplies interchangeability across the Win32 platforms.

Multiplatform portability, however, does not figure prominently in Microsoft's marketing plans. In the past, Microsoft has made noises about multiplatform support, but has rarely followed through. It is very likely that .NET closes the door on portability outside of Win32.

7. .NET CONTAINS A RAD PLATFORM

Microsoft is quick to claim this, and by some comparisons it is true. The Visual Studio .NET IDE nicely ties all the pieces together and has code generation capabilities.

When drawing a comparison between the RAD development features of .NET and PowerBuilder, Microsoft's rapid development claim loses some steam. .NET could be called a pure programmer's programming environment. It delivers exceptionally fine control over the smallest details by exposing them. This is very pleasing to a programmer who desires to write deep and elegant programs and believes they should be in charge of the finest details because they can do a better job of implementing them. This ability to build a watch to tell the time is useful in certain situations, but it is not the definition of RAD.

Rapid Application Development means the development tool paves the way for development; it means reaching the goal of a functional and robust application in the shortest amount of time. Once the design is specified the developer should not have to look right nor left and the development tool should take care of the details so the programmer can concentrate on the unique aspects of the application. The programmer should be focusing on exposing the necessary portions of the database structure to the user interface so the workflow best supports the end user, or focusing on the business rules without being overly concerned with the underlying implementation used to store them.

It is important to remember that .NET is a cluster of technologies that represent a product strategy. Microsoft did not win the Java wars and could not dominate the Java development market. It was no surprise when Visual Studio dropped support for Java in the .NET version. Microsoft unabashedly promotes its 'JUMP to .NET' strategy and offers its Java Language Conversion Assistant to convert Java to C#. Rather than promoting JSP development, Microsoft explains at great length why developers should be using ASP.NET. This seems to demonstrate an underlying product strategy of Microsoft-only domination, rather than an open platform affording a best-of-breed selection of development tools.

Suspicion of this world domination strategy is one of the reasons .NET has not lived up to the company's early expectations of wildfire adoption. Assimilation by a single-vendor, regardless of the vendor, makes planners pause for all the right reasons. These reasons include future price hikes, exposure to underlying flaws like security holes and virus susceptibility with little recourse, and the potential inability to use emerging technologies from other vendors.

ARGUMENTS AGAINST A .NET MIGRATION

Now let's examine some reasons why a wholesale conversion from PowerBuilder to Microsoft .NET might not be the best course of action:

1. EXISTING CODE AND APPLICATIONS REPRESENT A SERIOUS INVESTMENT

Mature organizations have huge investments in their existing applications. The code has been developed, tested, deployed, successively refined, and proven. The development staff has been trained and understands not only the body of developed code, but also the way the code maps to the problem domain. For companies with PowerBuilder applications, this code represents a large, multi-year investment in code development and programmer expertise. Unless a development platform is discontinued, or it is unable to keep pace with new initiatives—neither of which is the case with PowerBuilder—there is no compelling reason to completely reboot a development organization in favor of .NET.

2. A RAPID APPLICATION DEVELOPMENT ENVIRONMENT GETS RESULTS

The confluence of the PowerBuilder IDE, PowerScript, PBNI, and the DataWindow makes PowerBuilder an excellent RAD tool. There is a reason this large body of PowerBuilder-based code exists—for Rapid Application Development of functional database-driven solutions PowerBuilder has no peer. The payoff for the development effort lies in the power of these applications to let users view and manipulate complex datasets without being unnecessarily concerned with the implementation details. Database-driven applications are hardly unique to

PowerBuilder, what is unique to PowerBuilder is the relatively low level of development effort required to build them. For an organization, RAD provides faster time to market with a quality product. RAD also means lowered development and maintenance costs because fewer programmers are able to accomplish more work.

3. SIMPLICITY STREAMLINES THE DEVELOPMENT PROCESS

PowerBuilder is a powerful Object Oriented (OO) development environment that largely hides the complexities of OO programming and the impressive set of features it offers developers. The PowerBuilder product itself is written C++ so it allows developers to have easy access to low-level functionality. Yet, the PowerBuilder development environment is user-friendly and easy to use, hiding an enormous level of implementation and portability details. While the product itself is rich and complex, developers are provided with an easy to use IDE for streamlined application development. A RAD 4GL environment means a wider variety of developers can use the tool. An individual with an aptitude for programming, but lacking formal training, can easily pick up the tool; this is less true of a 3GL environment.

4. DATAWINDOW FUNCTIONALITY IS NOT AVAILABLE ELSEWHERE

Why is the PowerBuilder DataWindow so popular? Because the DataWindow is a powerful database visualization tool packaged in an easy to use interface. Programmers who have used the DataGrid control from Microsoft know it can be a painful programming experience because it requires the developer to program to an extremely detailed programming interface. Efforts to use it in even slightly unconventional arrangements quickly devolve into cell-by-cell programming. Even more unfortunate is the DataGrid's limited reporting capability.

The DataWindow and the Web DataWindow are now XML-enabled for importing and exporting XML and optionally applying formatting with XSL. DataWindow's underlying constructs present complex information using a graphical interface with little programming required.

5. HARDWARE/OS PORTABILITY KEEPS FUTURE OPTIONS OPEN

PowerBuilder supports the Win32 platforms and, PowerBuilder components can be deployed to application servers on Unix platforms, including Sun, HP, and IBM AIX. PowerBuilder leaves cross-platform development as a current and future option and does not lock PowerBuilder applications out of Unix environments.

PEACEFUL COEXISTENCE

The issue of .NET versus PowerBuilder does not have to be an either/or proposition. Sybase recognizes the value of maintaining an open development environment, while also recognizing the objective draw of .NET once the marketing hype is stripped away. .NET is one of the currently viable development platforms and integrates Win32 operating systems, services, and Microsoft Office technology. .NET support is a requirement for software vendors. Sybase PowerBuilder supports Microsoft .NET and will continue to enhance this support so PowerBuilder shops can continue leveraging their investment. Some time ago, Sybase developed a four-phase implementation strategy to support .NET. Sybase continues to execute to that carefully considered approach.

Phase I – This phase was completed with the release of PowerBuilder 9.0 and support for Web Services. PowerBuilder applications can build, publish and consume Web Services for Microsoft .NET and other frameworks without extensive knowledge of Simple Object Access Protocol (SOAP) or Web services Definition Language (WSDL).

Phase II – This phase is underway with the beta release of DataWindow.NET. Phase II introduces DataWindow.NET and DataStore.NET. DataWindow.NET is a separately packaged tool for developers using other .NET languages to take advantage of the patented DataWindow technology for data access and manipulation.

Phase III – To enable developers to reach into the .NET class libraries and the rest of the platform, this third phase will implement a compiler that produces Microsoft Intermediate Language code. This code will execute as managed code in the Common Language Runtime. This means PowerBuilder applications can be intermingled with other .NET applications.

Phase IV – This final phase opens the PowerBuilder IDE to support the .NET User Interface designer. This will allow PowerBuilder applications to implement .NET user interface components.

Sybase is executing to this plan; phase I and a portion of phase II are completed. In retrospect, given the slower than expected rate of .NET adoption, the plan exhibits a high degree of vision. Rather than an all or nothing push to .NET, Sybase put its efforts into making PowerBuilder better in ways that would serve all customers, including the .NET community. Sybase has taken, and continues to take, a thoughtful approach to its .NET support. PowerBuilder is good for building applications that are mission critical and data driven. With Phase I completed, and as this .NET plan continues to unfold, PowerBuilder applications can productively drive the database engines of .NET solutions.

CONCLUSION

Objectively, although not the panacea it aspired to be four years ago, Microsoft's .NET technology is impressive. It is the fruit of some of the best minds in the business, working for a dominant company with a powerhouse-marketing arm. It has a referential integrity that knits together the diverse pieces of Microsoft's multi-front software organization.

PowerBuilder was designed for database-driven applications and has succeeded in virtually all classes of business applications that require data access, manipulation, and processing. The PowerBuilder IDE is powerful, simple, and intuitive. Using the IDE to create production PowerBuilder applications represents the essence of RAD. As demonstrated by its track record, Sybase is committed to continue evolving PowerBuilder to address new technologies. PowerBuilder has added XML, DOM, JSP, and Web Services support, among others, over the last few releases. These enhancements represent an open strategy that supports developers as they continue expanding the utility of existing applications into arenas that include .NET and affordably add value to their technology investments.

Sybase Incorporated
Worldwide Headquarters
One Sybase Drive
Dublin CA, 94568 USA
T 1.800.8.SYBASE
www.sybase.com