SYBASE®

SYBASE® POWERBUILDER® VERSUS JAVA™

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.

SYBASE
POWERBUILDER

INTRODUCTION

While the passage of time adds wrinkles and thins hair, it also provides the ability to examine the interrelationships and relative importance of past events in a broader context. There has been a long-running debate in the PowerBuilder community between choosing PowerBuilder or Java for application development. In the late 1990s Java gained immense popularity within the programming community, especially compared to that of C++. The Java language represented the next evolutionary step in Object Oriented development, offering simplified portability and clear advances in memory management. There was an immediate land rush to move to Java development; almost overnight, Java books appeared at local bookstores and experts began offering training programs. From today's vantage point, Java has matured. Thousands of Java-based development projects have completed full development life cycles and the debate—PowerBuilder vs. Java—can be re-examined in a richer retrospective context.

This whitepaper addresses the issues faced by development organizations considering either a transition from PowerBuilder to Java for their legacy code, or deciding between PowerBuilder and Java for new development. It examines the original promise of Java and where it has been successfully applied; and also examines considerations that need to be kept in mind when making a decision between Java and PowerBuilder.

THE JAVA EXPLOSION

Java exploded on the scene in the mid-1990s, rapidly meeting the pent-up demand for a new Object Oriented third generation language (3GL) capable of competing with C++. It helps to look at the historical context to understand why Java generated such excitement. When Java was introduced in 1995, Object Oriented technology had matured into a popular programming discipline. Many organizations had been using C++ for a number of years and programmers had made the conceptual transition to Object Oriented architectures and development. Object Oriented design patterns had become popular because they made sense to C++ developers and reflected a shared understanding in the OO thinking process.

Despite the beauty of Object Oriented development, C++ programmers also knew the frustrations of memory leaks and the headaches associated with multiple inheritance. This new language of Java embodied the collective gestalt of Object Oriented development, without the issues and frustrations associated with C++. Java had OO elegance—without the baggage. Additionally, Java was designed to be highly portable and carried the slogan of "Write Once, Run Anywhere." Java was released at a time when Linux was starting to gain momentum and many programmers were concerned about Microsoft's dominance in the marketplace. Java was a language that not only offered portability beyond Microsoft, but it was a development tool that did not originate in Redmond, Washington. 1995 was also the year that many developers became aware of the Internet, and Java showed promise as language for this new medium. Sun Microsystems was unprepared for the interest in its new language and the unexpectedly high volume of Beta version downloads crashed its server. Clearly, Java's introduction was timely.

The 1995 introduction of Java and the 1997 release of the Java Development Kit 1.1 generated ripples of insecurity in PowerBuilder shops. If Java was the juggernaut the trade press was touting, then was PowerBuilder approaching obsolescence? PowerBuilder at this time did not have many of the advanced features of today's PowerBuilder, most of which were added to track with the unfolding Internet revolution. At the time Java fairly sparkled with promise and PowerBuilder, by comparison, looked stodgy—like your father's Oldsmobile. PowerBuilder had proved itself as a Rapid Application Development system in comparison to C and C++, but Java looked to be something altogether different. Viewed in the context of the time, the concern among PowerBuilder shops was not unreasonable.

ABOUT JAVA

Java is a third-generation language (3GL) that was originally designed for programming smart appliances. Java is Object Oriented and uses single inheritance, rather than the multiple inheritance of C++. Syntactically the language is similar to C++, but unlike C++, Java compiles down to a portable, architecture-neutral byte code.

Nearly a decade after its introduction, Java has matured and found its niche primarily in the Java 2 platform, Enterprise Edition (J2EE). Java has grown with the Internet, is deployed on a significant number of application servers, and is proven as a tool for developing distributed applications that function on multiple platforms. By contrast, the Java 2 Platform, Standard Edition (J2SE) version of Java used for desktop application development has struggled because of complexity and poor user interface performance. J2SE has significantly lagged behind J2EE's popularity.

ABOUT POWERBUILDER

PowerBuilder is Sybase's Rapid Application Development (RAD) system used to build database-driven client/server and desktop applications. PowerBuilder is a fourth generation language (4GL), which means it manages and abstracts the underlying details that are not directly related to building the application. PowerBuilder is Object Oriented and its intuitive IDE complements the inherent productivity of the environment.

Like Java, PowerBuilder too has matured in the last decade. Internet technology support, standards support, and extended connectivity to dissimilar systems gives PowerBuilder additional muscle in the new millennium. Sybase has carefully and deliberately evolved PowerBuilder to be a technology that both supports, and reflects, the current state of the enterprise. Today's PowerBuilder has much of the depth associated with third generation languages, while maintaining and improving the ease-of-use and RAD functionality from its 4GL heritage.

JAVA COMPLEXITY

Over the years, several themes emerge repeatedly for Java's inability to truly compete in the PowerBuilder's main arena of desktop and client/server applications. Some reasons are inherent in Java, such as 3GL versus 4GL and weak user interface support. Other reasons, paradoxically, are a consequence of Java's overall success. The reasons can be distilled down to one basic truth: Java is more complex than PowerBuilder and, consequently, harder to use. Java requires more code, is much harder to learn, and its popularity has fostered a high degree of market fragmentation among competing vendors. The following section examines Java's complexity in more detail.

LINES OF CODE NECESSARY TO ACHIEVE FUNCTIONALITY

A short-lived measure of programmer productivity was lines of code written over time. For obvious reasons, this measure did not last long. In a Dilbert comic strip the team was told they would be paid for the lines of code they wrote; Wally exclaims, "Wahoo! I'm gonna write me a minivan!" A far more significant measure of modern productivity is functionality completed over time. A lower-level language provides a finer degree of control, but it also requires more lines of code to complete a task.

As a 3GL, Java, like C#, C++, and C needs more lines of code to provide equivalent functionality than does a 4GL environment like PowerBuilder. From the point of view of productivity, more lines of code are undesirable. In application development, the more lines of code written, the longer it takes to write the code. This is a self-evident concept, but it is important to remember the basics when evaluating a development platform. Development is not just writing code, it means writing working code. When writing many lines of code, a developer has more opportunity to introduce mistakes into the code, which requires additional debugging time. Fewer lines of code also means long-term maintenance will be a more productive process. Another consideration is the burden of the learning curve when adding new members to a team; the more code a new developer has to understand, the longer it will take them to be productive. This is not to say that developers should be writing as few of lines of code as possible, rather that the lines of code they write should equate to more application functionality. Programming should not be an exercise in the beauty of writing code; it should be a directed process of putting robust functionality into the hands of users.

COOPERATIVE ANARCHY

A productivity comparison between PowerBuilder and Java transcends the lines of code to functionality ratio. Perhaps one of the most difficult areas of Java development is keeping abreast of new frontiers that are constantly in motion, all in different directions. The number of Java development options means more choices need to be made at the beginning of development so it can take longer to get started with Java projects. Some Java frameworks, like J2EE and EJB, have exploded in like measures of functionality and complexity; even an experienced Java developer needs to work hard to stay current with the various technologies.

Java, unlike PowerBuilder or C#, does not evolve to a master plan from a single corporate entity. There are a large number of smart, connected, and yet fiercely independent Java programmers in the workplace today. This community of developers is constantly generating competing solutions and add-on technology; unsurprisingly, the solutions often diverge in their implementation and functionality. This cooperative anarchy of vendors and developers lacks an overarching corporate vision and this is one of the reasons contributing to Java's complexity. Most languages/development environments have a few ways to skin a particular cat, but Java can offer a bewildering number of approaches to a single class of problem. This large number of competing vendors has created a fragmented market of tool choices from IDEs to libraries. For example, in a typical local Java User Group meeting, the first half is a presentation on a subject such as Web Services and the second half is a spirited discussion on the merits of competing vendor Web Services technologies. No one at the meeting questions why it is this way; it is simply the fruits of communal development.

STEEP LEARNING CURVE

Learning Java seems like it should be easy; there are many books for beginners. It is not too difficult for an experienced programmer to get a simple application up and running in Java. Once that initial hill is conquered, Java becomes complex. Learning the Java language is relatively easy and requires practice; learning and mastering the breadth of Java technology is very difficult and requires passion. The many facets of Java create a steep learning curve and development organizations that move to Java often find the environment to be more complicated and expensive than originally anticipated. When developing in Java, there are times when the language seems determined to keep a developer from doing things simply.

There is a subtle cachet of assumed brilliance among Java developers. When working with a community of Java developers it can be disheartening for an average programmer to operate at the expected level of the elite. The language is pure and beautiful and a developer is expected to be able to extrapolate multiple solution sets from the underlying elegance in the language; it is as if given a brick, one should be able to imagine a cathedral. The Java development community has a large number of purists who are not looking for a rapid application development environment because they are more than capable of doing it themselves.

USER INTERFACE SUPPORT

Database-driven applications have two main components: the database connectivity/manipulation code and the user interface. The Java-based Swing user interface tool kit is used to develop applications with the look and feel of Microsoft Windows. The Swing programming interface is surprisingly difficult to use and is one of the first stumbling blocks that trips up new Java developers. It is easy enough to lay out the Swing components with an IDE, but it becomes tricky to write the manipulation code for the components. For instance, wiring a database into a JTree component to make it database aware seems far more difficult than it needs to be.

Java's complexity has been a two-edged sword; its power appeals to serious systems programmers, but the level of detail has hindered adoption among developers who want a productive application development system. This dichotomy, and the process of natural selection, allowed Java to exploit the J2EE application server niche while largely missing the desktop market.

COMPARING JAVA AND POWERBUILDER

Drawing comparisons between Java and PowerBuilder reveals stark contrasts and an interesting pattern emerges; Java has done very well in the arena of application servers, yet sputtered in the client/server and desktop application market where PowerBuilder excels. Development organizations contemplating a conversion from PowerBuilder to Java should consider using both languages, each for its own strength. The multi-year debate on the merits of PowerBuilder versus Java includes several recurring themes that are worthy of examining in greater detail:

JAVA REPRESENTS A NEWER APPROACH TO CODE DEVELOPMENT

Java is a newer language than PowerBuilder. Newer does not necessarily mean it is a better choice for development. Instead, the development task should be evaluated for the applicability of Java or PowerBuilder. If the task is populating new business rules on an application server, and the development staff is comfortable using Java, then Java is a good candidate for this development. If the task is building a rich client or smart client application, then PowerBuilder is the best choice because of its unmatched productivity in building this class of applications.

JAVA DOES NOT HAVE POWERBUILDER'S DATAWINDOW®

Java's Swing user interface is a serious weakness in J2SE. Java's Swing has a JTable, but its effectiveness cannot match PowerBuilder's DataWindow. JTable is a row/column cell oriented control that requires serious coding to perform rudimentary tasks. PowerBuilder's DataWindow has extreme data manipulation capabilities, advanced on-board status tracking, and intuitive setup; Sybase's DataWindow blows the doors off anything else in the Java user interface.

JAVA HAS MORE FEATURES AND THIRD-PARTY ADD-ONS AVAILABLE

Java has an entire development community building new functionality and making it available to other Java developers. Without a master vision, cooperative anarchy is the guiding force. The very number of add-ons can dilute a programmer's effectiveness because choices need to be made and the utility of the various options need to be evaluated, making it harder to forge ahead in a project. A smaller, but very capable toolset increases productivity.

POWERBUILDER HAS BETTER EVENT HANDLING

PowerBuilder was designed to support event-driven applications. The events are easy to identify and respond to in code. To manage event handling in a Java application, handling code must be written to instantiate and initialize the listener classes that administer the events. Again, code needs to overtly identify and trap the events, and finally, the actions taken in response to the events also need to be coded.

JAVA DEVELOPERS ARE MORE PLENTIFUL THAN POWERBUILDER DEVELOPERS

There are more Java programmers than PowerBuilder programmers. Java developers are easier to add to a project or replace in a project than PowerBuilder developers. However, it is easier to train a PowerBuilder developer than a Java developer. If the target application domain is database-driven applications, then should the course of development be based on the availability of programmers who are skilled in using an inefficient development environment? PowerBuilder development achieves more results with fewer programmers, which lowers overall project costs.

Another point needs to be made here, and that is that there is certainly no shortage of PowerBuilder developers. There are literally hundreds of thousands of PowerBuilder developers in the workplace today. Some of these may have experience on older versions of PowerBuilder, but they have the relevant skill set necessary for staffing or backfilling PowerBuilder projects.

JAVA HAS A PURER SYNTACTICAL ELEGANCE

Java does have a syntactical elegance that is perhaps only matched by C#-. This is a draw for programmers, especially for formally trained programming purists. Again, the goal needs to be examined. Is the goal to write exceptionally pretty code, or to develop applications rapidly? If code elegance is desired, than by all means Java is a good candidate. If the goal is to quickly get robust applications to market, then PowerBuilder is a better choice. In the time it takes for a team of Java developers to build a pleasing object diagram, a smaller team of PowerBuilder developers can often have a functioning application developed and deployed. Outside of an academic environment, coding is not a beauty contest; in a commercial environment, building stable functionality in the least amount of time is the criteria for success.

POWERBUILDER'S USER INTERFACE IS SUPERIOR

Java's Swing user interface was built as an addition to Java's Advanced Widget Toolkit (AWT). Both Swing and the AWT are slow, difficult to program, and have a look and feel that, especially on the Microsoft platforms, is bothersome to users. The AWT relies on Java primitives for drawing and is consequently slow. Depending on the speed of the machine and the complexity of the user interface, the window can feel sticky or unresponsive to the mouse because events stack up in the event queue. The Swing programming interface requires—at least from the PowerBuilder perspective—an enormous amount of code for even simple tasks. For example, Swing and Java lack native functionality that is analogous to PowerBuilder's EditMask for field validation as the user types. Either the programmer must write more code, or allow the user to make more input mistakes that need to be trapped further downstream in the workflow. This is an example of Java's productivity tradeoff between development time and application functionality.

Looking back after years of Java deployment in the field, the user interface stands out as a primary cause of Java's lukewarm adoption in the desktop and client/server markets. Java developers had to repeatedly dash themselves against the rocky user interface before they gave up and went off to make magic with Web application servers. Many PowerBuilder shops that made the early jump to Java, and found themselves mired in user interface and database connectivity details, with a precipitous fall in productivity, have been migrating back to PowerBuilder.

POWERBUILDER TODAY

With the release of PowerBuilder 9.0 a PowerBuilder application is able to connect to Java EJBs on J2EE servers. Now developers have the capability of using the strengths of PowerBuilder with its graphical front end to access distributed business logic written in Java on J2EE application servers like EAServer, BEA, and IBM to get the best of both worlds.

PowerBuilder release 10.0 continues PowerBuilder's record of enhancements to support the latest technologies. PowerBuilder 10.0 includes:

- **PowerDesigner® plug-in for OO modeling** – This plug-in gives PowerBuilder programmers the ability to model code in the IDE, generate code, reverse engineer existing applications, and perform round trip engineering.
- **XML Web DataWindow** – This version of the DataWindow contains performance enhancements and separates the content, layout, and style of the DataWindow on the Web. Using Cascading Style Sheets, the presentation can be customized and targeted for specific needs including Section 508 accessibility. To support the XML Web DataWindow, the DataWindow Painter now supports XHTML customization.
- **UDDI registry browsing** – Web Services are again gaining momentum and PowerBuilder 10.0 supports browsing UDDI registries to locate Web Services. PowerBuilder already supported consuming and publishing Web Services, and the UDDI search capability further ties PowerBuilder's Web Services offerings together.
- **Unicode enablement and support** – Multiple languages can be displayed on a page at the same time. This feature is very useful for interface layout in multiple languages. The Unicode support streamlines multi-language development and opens new markets to PowerBuilder applications.

PowerBuilder has not stood still while Java matured; clearly Sybase has continued to allocate development resources to PowerBuilder. The PowerBuilder of today is exceptionally productive with rich support for a broad spectrum of technologies. Java no longer looks shiny new, and PowerBuilder most definitely does not look stodgy in comparison.

CONCLUSION

Now that the dust has settled and the initial wave of Java excitement has passed, and zealous programmers have thoroughly explored both its possibilities and its limitations, it is possible to look at the debate between PowerBuilder and Java with a discernment gained from several years of actual implementations. Java is defined; certainly it will continue to evolve, but only within the confines of its definition, as the inertia of legacy code acts as a restraint against radical change. This fulfillment of time makes it possible to analyze Java for what it truly has become, not for the future possibilities that were turning heads in the late 1990s. The analysis reveals that Java has been a boon to systems programmers working on application servers. Java has played a significant role in the growth of the Internet by simplifying low-level Object Oriented programming. Java's slower execution speed is the only thing that keeps it from completely replacing C++. The analysis also reveals where Java has not realized its perceived potential. Java has been disappointing in the client/server and desktop market; it has not been able to compete because of user interface performance, execution speed, ease of development, and the dominance of 4GL systems like Sybase PowerBuilder.

PowerBuilder was designed to be a 4GL Rapid Application Development tool for client/server and desktop application development, the very areas where Java has foundered. In that marketplace PowerBuilder has succeeded and continues to succeed. Unlike Java, Sybase has a corporate vision that directs the evolution of PowerBuilder. Far from being moribund, today's PowerBuilder is a dynamic and productive tool that also performs very well as the presentation front end to application servers.

The long-running debate concerning the selection of Java or PowerBuilder as a development tool concerns two very different development systems with very little overlap in their effective markets. Rather than polarizing the debate, Sybase continues to blur the lines by increasing PowerBuilder's Java support. There is an old engineering adage that says, "Defining the problem is half the solution." The process of selecting the best tool for the job means understanding the task. If the task is adding new business logic on a J2EE application server with a Java-trained staff, then Java wins. If the task is to build database-driven applications for a desktop or client/server implementations, or to Web enable existing PowerBuilder code, then PowerBuilder is a clear winner over Java.

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